Universidad Carlos III de Madrid
Departamento de Informática

Doctorado en Ciencia y Tecnología Informática

EXTRACCIÓN AUTOMÁTICA DE MODELOS UML CONTENIDOS EN IMÁGENES

Tesis Doctoral

Autor
Valentín Moreno Pelayo

Director
Prof. Dr. D. Juan Bautista Llorens Morillo
Prof. Dr. Gonzalo Génova Fuster

Leganés, noviembre de 2015

TESIS DOCTORAL

EXTRACCIÓN AUTOMÁTICAS DE MODELOS UML CONTENIDOS EN IMÁGENES

Autor: Valentín Moreno Pelayo

Director/es: Juan Bautista Llorens Morillo
Gonzalo Génova Fuster

Firma del Tribunal Calificador:

Firma

Presidente: (Nombre y apellidos)

Vocal: (Nombre y apellidos)

Secretario: (Nombre y apellidos)

Calificación:

Leganés, de de

A mi familia y amigos

Resumen

Aunque parezca extraño, pese a no poder encontrar sitios web especializados en ofertar diseños de software representados mediante diagramas UML, existe una ingente cantidad de documentación a disposición de cualquiera, y que contiene dichos modelos: **como imágenes** en documentos textuales. Este universo de información no se encuentra fácilmente accesible para los desarrolladores porque no es posible, con la tecnología actual, buscar de forma precisa información semántica dentro de imágenes. Lo único que pueden hacer los desarrolladores es intentar buscar documentos relevantes, leerlos, y decidir si los diseños le sirven a sus intereses.

Para evitar este problema, y conseguir poner a disposición de toda la comunidad de desarrolladores centenas de miles de diseños, este trabajo pretende desarrollar la metodología necesaria para poder extraer la información textual y gráfica de las imágenes que representen diagramas UML, y convertirla en información pura UML (es decir, en modelos UML reales).

El poner a disposición de los analistas, desarrolladores de software, o interesados tal cantidad de diagramas y modelos de software permitirá la aplicación de técnicas modernas de reutilización de software basadas en la búsqueda de diagramas UML. La búsqueda de diagramas UML de todo tipo (estáticos, dinámicos, arquitecturales, de Casos de Uso etc.) mediante similitud a uno dado permitirá potenciar los desarrollos de software de calidad, controlados en el coste, y en el tiempo de desarrollo: las tres virtudes de la reutilización de software.

La complejidad de esta propuesta radica en muchos aspectos, todos ellos entrelazados: por un lado hay que considerar que la información de partida se encuentra representada con diferentes tipos de calidad, mediante bits de colores o tonos de grises. Por otro lado su semántica viene dada por la combinación de texto en lenguaje natural y estructuras gráficas. Estas estructuras gráficas tienen asociada una información semántica, accesible a la interpretación humana, que depende del tipo de diagrama. Los diagramas que representan diseños de software son documentos en formato visual con alta estructuración y contenido semántico, que se deben distinguir unos de otros. Debido a su formato en forma de imagen requieren un preprocesado mediante técnicas de visión artificial, OCR y técnicas de clusterización o clasificación basadas en aprendizaje automático.

Precisamente este será el principal cometido de esta tesis: la extracción de la semántica de los diagramas en forma de imágenes encontrados en la web. La información obtenida de estos diagramas, principalmente UML, debe incluir información textual e información estructural. A la información textual se obtendrá mediante técnicas de OCR mientras que la información estructurada será detectada mediante reconocimiento de formas combinado con Inteligencia Artificial.

El resultado de esta propuesta sería una metodología que podría ser aplicada para cargar repositorios de diagramas UML a partir de imágenes existentes en internet, con vistas a su posterior aplicación y puesta a disposición de los usuarios: un GOOGLE de diagramas UML.

Summary

There are many interesting sites in the web offering reuse of source code, but no one giving the choice to identify, find and reuse design models using UML. However, even if this data seems to be sad, a simple web search can give you astonishing results:
Get into GOOGLE images and search for "UML Class diagram". Thousands of images will suddenly be available for you. The bad news: they are images. You cannot find anything on them, you cannot find them by content. You cannot compare them. You can, simple, download them.

Could you be interested in working with those images, finding similar ones, etc.? In order to solve this problem, and reach hundreds of thousands of UML designs, this work intends to develop the necessary methodology to extract the textual and graphical information contented in UML based images, and convert them in, exactly, UML information (real UML Models represented in a UML object model)

The possibility to offer such amount of diagrams to software analysts, software developers, or simply interested stakeholders will allow them to apply real, systematic and modern software reuse based on UML diagrams information retrieval. The possibility to find all kinds of diagrams (static, dynamic, architectural, Use Case, etc.) by similar content will strengthen software development based on the best quality, controlled cost and time to market principles: the three real benefits of Software reuse.

This proposal has several difficulties in different fronts: to start with, one must consider that all the information is usually stored in low resolution images, where texts are difficult

to read and understand and boxes and arrows are not properly drawn. And, on the other side, the semantics comes from the combination of text represented in Natural Language and graphical structures. These structures have associated semantic information, understandable by humans, which depend and change with the diagram types. Due to these problems, Artificial Vision, OCR, classification and automatic learning algorithm must be used in this thesis.

This thesis, therefore, will attempt to extract semantic information for images representing UML Diagrams found in the web. The information extracted will be both textual and graphical. OCR technology (existing already) would be used to textual information. In order to extract graphical information a semantic model combined with AI will be used.

The result of the proposal will be a methodology that will allow repositories (in the web or private) to offer UML diagrams based on (and pointing to) images found in the web, for further reuse.

Agradecimientos

Es difícil por no decir imposible manifestar mi agradecimiento a cada una de las personas presentes y significativas en mi vida. Y digo en mi vida ya que no sería correcto circunscribirme únicamente a aquellas que han contribuido más directamente en la conclusión de este trabajo. Por ello, para que nadie pueda sentir mi olvido, hago alusiones en muchos casos de forma genérica.

Expreso mi gratitud a:

- Juan Llorens, mi director de tesis y ejemplo a seguir. Sin ninguna duda la persona que más admiración profesional me despierta.
- Gonzalo Génova, codirector de esta tesis, investigador completo en todos los sentidos. Siempre es un placer compartir con él cualquier actividad intelectual.
- Todos los miembros del grupo de investigación *Knowledge Reuse* y al personal relacionado con el mismo. Grandes compañeros que me hacen disfrutar mi profesión. Me siento afortunado por pertenecer a este grupo.
- Mi familia, razón por la que existo y merece la pena existir.
- Mis amigos, algunos ya incluidos en los roles anteriores, en cualquier circunstancia encuentro en ellos apoyo incondicional y sosiego permanente.
- Manoli, por su ayuda inestimable y desinteresada.
- Ana y Merce por estar presentes en todo momento.

Tabla de Contenidos

Índice de Ilustraciones

Índice de Tablas

Capítulo I: Introducción

1.1 Motivación

La Reutilización de Software es una rama de la Ingeniería Informática que estudia métodos para mejorar la calidad, el coste y el tiempo de desarrollo de sistemas informáticos intensivos en Software, mediante la búsqueda y aplicación de los resultados realizados en desarrollos de software anteriores [Frakes et al., 1998]. Sus fundamentos se basan en la noción de que la gran mayoría de los retos (problemas, soluciones, códigos,…) que debe solucionar un equipo de desarrollo informático ya han sido resueltos por colegas a lo largo del planeta, de una u otra manera. En estos casos, la reutilización de software plantea como metodología la utilización de artefactos software existentes y verificados en su calidad y rendimiento en la construcción de nuevos productos software. Estos artefactos pueden ser de cualquier tipo: código fuente, diseños de Software, módulos, especificaciones, arquitecturas, pruebas específicas, suites de pruebas, documentación, estimaciones, etc. Obviamente, el principal problema de esta forma de entender el desarrollo de software se centra en cómo conseguir que todos los artefactos reutilizables sean encontrados por el sistema informático que ofrezca reutilización, y cómo conseguir que, a su vez, el candidato reutilizador los encuentre. En suma, se trata de resolver problemas de identificación y captura de artefactos reutilizables (para cargar el sistema de reutilización (repositorio)) y recuperación de información para que los ingenieros pueda encontrarlos.

Es de sobra conocido en el ámbito de la investigación en "Software reuse" que la reutilización de artefactos (Activos) en fases más abstractas (de alto nivel) del ciclo de vida, como pueden ser los diseños (diagramas funcionalidades, estructurales (UML) o

dinámicos (UML)) resulta más provechosa que la de artefactos de otras etapas inferiores como es el código fuente [Boehm, 1988] [Dutton, 1997]. Esto se debe en gran medida a que los artefactos de bajo nivel quedan desfasados mucho más rápidamente que los de alto nivel, donde el problema que representan suele perdurar mucho más tiempo como relevante [Booch et al., 2005]. Adicionalmente, unos de los motores de la reutilización es la trazabilidad, que se ve favorecida cuando los componentes seleccionados se encuentran más arriba en la escala de nivel de abstracción del ciclo de vida.

La documentación correspondiente a estas etapas de alto nivel es por tanto muy valorada en el ámbito de la reutilización de software, y en particular los diagramas UML (de todo tipo). La razón de este interés, se puede encontrar seguramente en que su definición es muy precisa (meta-modelo de UML claramente definido [Booch et al., 1999], y su estructura y contenido resultan muy apropiados para este tipo de tareas, con una representación muy claramente reutilizable. Sin embargo, para que un entorno de reuso sea aceptado por la comunidad de desarrollo (tanto la denominada abierta como las más profesionales) es necesario disponer de una amplia colección de este tipo de documentación con el fin de seleccionar los resultados más afines al proyecto software que se desea desarrollar. Pese a que el diseño de sistemas informáticos ya es una de las actividades claramente maduras en el proceso de desarrollo de Software actual, no se ha conseguido aun un entorno de reuso que permita su aplicación en sistemas informáticos diferentes de los que definieron el diseño. En la actualidad, toda la reutilización que se realiza de diseños informáticos se basa en el hecho de que el re-usador conoce de antemano lo que desea reutilizar (Reutilización ad-hoc, [Karlsson, 1995]). Para complicar aún más la situación, no es normal a día de hoy que los diseñadores que actualmente publican su código de forma libre y abierta, publiquen también sus diseños correspondientes en formatos electrónicamente entendibles (como un CASE o similar). Sin embargo, pese a que resulte contradictorio, la frase anterior no es cierta del todo. Una enorme mayoría de diseñadores sí publican sus diseños informáticos en abierto: **COMO IMÁGENES.** Existe una cantidad ingente de diseños de software accesibles por la comunidad informática, pero en formato PDF, o documento, mediante imágenes.

En este sentido, se plantea la idea de construir repositorios de diseños de software recopilando de la Web cualquier documento que represente un diseño en UML, de forma que la tecnología desarrollada en esta investigación consiga extraer la información semántica (el diagrama UML existente) de la imagen, almacenándola como información

UML. Esto permitirá su búsqueda mediante consultas semánticas (por ejemplo mediante el dibujo de un pequeño diagrama UML en el buscador) para que el sistema informático pueda buscar diseños similares.

En concreto en esta tesis se desea resolver el paso de imagen binaria a información UML estructurada. La tecnología adicional necesaria (web, crawlers, motor de búsqueda, algoritmos de búsqueda…) para completar la idea general expuesta ya se encuentra desarrollada por el grupo multidisciplinar de investigadores encabezado por el Dr. Juan Llorens.

1.2 Hipótesis

Mediante técnicas de visión e inteligencia artificial es posible establecer un método automático para identificar imágenes que representen diagramas software y extraer su información, incluso si la procedencia de las mismas es la Web.

1.3 Objeto

Proponer un método automático para la identificación y extracción de la información de representaciones de diagramas software en formato imagen con baja resolución.

1.4 Aplicación

La principal aplicación práctica es la construcción, con niveles mínimos de supervisión humana y a partir de consultas web, de repositorios de diagramas UML orientados a la reutilización del software.

1.5 Objetivos

El objetivo principal de la investigación es crear una metodología que sea capaz de extraer información de diagramas lógicos de UML a partir de información binaria de imágenes de manera automática. Además, la metodología incorporará técnicas de procesamiento de imágenes que permitan determinar cuáles de ellas contienen diagramas UML, y cuáles otro tipo de información.

Adicionalmente, forma parte de los objetivos el proporcionar algoritmos de procesamiento de imágenes que optimicen la aplicación de la metodología incluso con

imágenes descargadas de la Web y que por tanto tienen baja resolución y se consideran inapropiadas para aplicarles software de OCR u otro tipo de reconocimiento.

1.6 Metodología

Para cumplir los objetivos se han establecido las siguientes fases en la metodología:

i. Detección de los elementos gráficos presentes en imágenes de diagramas UML.
ii. Filtrado automático de imágenes que representan diagramas UML.
iii. Reconocimiento de los textos asociados a los elementos gráficos presentes en imágenes de diagramas UML.

El desarrollo de la investigación se estructura en varios bloques, asociados uno a uno a cada fase metodológica, que permiten cumplir el objetivo principal propuesto en la tesis. Los tres bloques comparten la misma estructura (Ilustración I-1).

- En la **fase de metodología** se expone, sin llegar al nivel de detalle, los pasos a seguir en la investigación para alcanzar el objetivo propuesto.
- En la **fase de desarrollo** se explica cada uno de los pasos expuestos en la metodología concretando cualquier particularidad relevante e incluyendo las decisiones que haya sido necesario adoptar.
- En la **fase de experimentación y resultados** se muestran cada uno de los experimentos y sus resultados correspondientes, incluyendo si es el caso los de evaluación.
- En la **fase de discusión y conclusiones** se presentan las conclusiones y los razonamientos que han conducido a ellas.

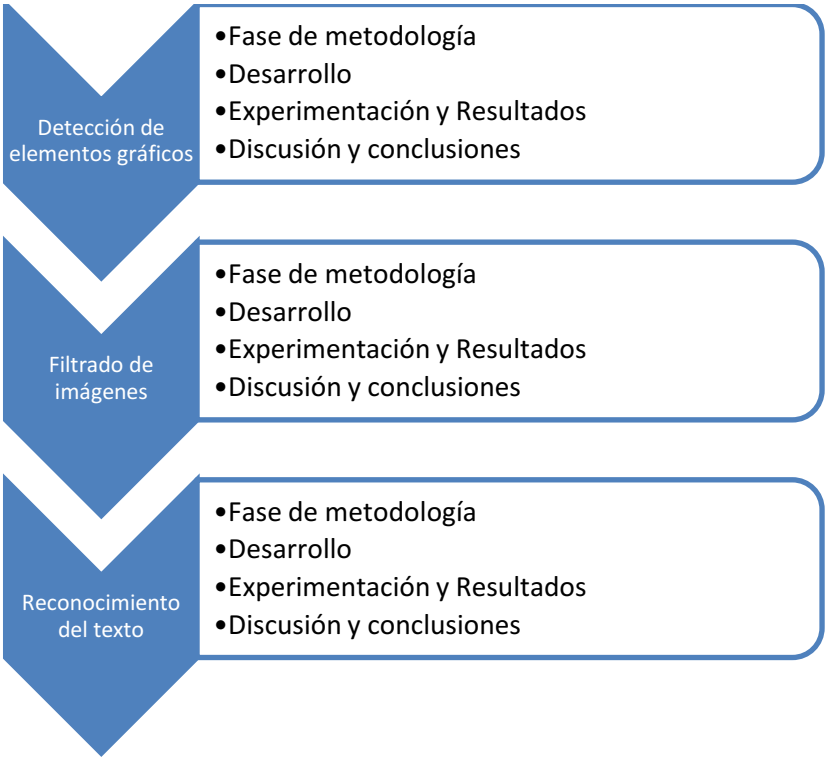


Ilustración I-1: Principales pasos de la metodología

1.7 Requisitos de la Investigación

En este punto se describen los requisitos de la investigación de acuerdo a los aspectos que delimitan el ámbito de estudio. Este estudio está restringido por requisitos temporales, por requisitos de las aplicaciones de software libre y por limitaciones establecidas en las pruebas efectuadas con el fin de que sean abordables.

Las implicaciones temporales de los procesos de investigación son en este campo determinantes debido al fuerte dinamismo de las tecnologías web. Los datos recogidos para la investigación se obtuvieron en el periodo (2009-2010), no obstante el proceso metodológico es adaptable a los posibles cambios de los algoritmos de los sistemas de recuperación. Por consiguiente debe tenerse en cuenta los periodos en los que se han realizado los análisis y los experimentos ya que algunas situaciones específicas habrán cambiado, aunque con la vigencia de las aportaciones metodológicas. En concreto, las imágenes descargadas de la Web, mediante consultas concretas en motores de búsqueda que solicitaban diagramas software, serían diferentes de repetirse esas demandas de información en otro momento. Por tanto, la proporción de imágenes descargadas que

realmente se corresponden con diagramas UML cambiará conforme evolucionan los algoritmos de los buscadores web pudiendo afectar a las evaluaciones de los filtradores de imágenes de diagramas.

Otra restricción temporal viene dada por el tiempo necesario para procesar una imagen con el fin de extraer su información. Dado que el volumen de imágenes procedentes de la Web podría ser muy elevado se considera apropiado limitar el tiempo medio de procesado de una imagen a un máximo de 20 segundos en máquinas de uso convencional.

Por otra parte, se ha priorizado las aplicaciones de software libre para los recursos de la investigación dado que no existe una financiación adicional que permita la compra de licencias de software (salvo las incluidas en el paquete de soluciones OFFICE proporcionado por Microsoft). El funcionamiento de las aplicaciones incorporadas está garantizado por su larga trayectoria en investigación, como la herramienta de análisis de datos Weka [Witten y Frank, 2005]. Otras, sin embargo, como el OCR MODI obtiene peores rendimientos que el de otras herramientas homologas comercializadas. No obstante, se mostraran comparativas de rendimiento con otros OCR comerciales con miras a futuras implantaciones en las que pudiese ser rentable invertir en ellas.

En cuanto a las limitaciones que se establecen de cara a acotar los experimentos dentro de lo razonable se presentan las siguientes:

- No considerar la extracción de información en imágenes de resoluciones inferiores a los 72 píxeles por pulgada
- Evaluar principalmente las herramientas software OCR más conocidas y utilizadas.
- No extraer información de imágenes en cualquier formato (Las imágenes de trabajo estarán en formato bmp u otros de los más populares). En particular se admitirán los presentes en la siguiente ilustración que presenta los formatos admisibles por la herramienta gratuita de procesamiento de imágenes XnView (http://www.xnview.com/).

BMP - Windows Bitmap
EMF - Windows Enhanced Metafile
GIF - CompuServe GIF
JPG - JPEG / JFIF
PCX - Zsoft Publisher's Paintbrush
PNG - Portable Network Graphics
PPM - Portable Pixmap
TIF - TIFF Revision 6

92I - TI Bitmap
CIN - Kodak Cineon
CSV - CSV
DCX - Zsoft Multi-page Paintbrush
DDS - Direct Draw Surface
DIS - DKB Ray-Tracer
DPX - DPX
FTS - Flexible Image Transport System
GRO - HP-48/49 GROB
HDR - ArcInfo Binary
HRU - HRU
ICO - Windows Icon
IFF - Amiga IFF
IFF - Explore (TDI) & Maya
IMG - Vivid Ray-Tracer
JIF - Jeff's Image Format
JP2 - JPEG-2000 JP2 File Format
JPC - JPEG-2000 Code Stream
KRO - Kolor Raw Format
MBM - Psion Series 5 Bitmap
MIF - Image Magick file
MTV - MTV Ray-Tracer
NGG - Nokia Group Graphics
NLM - Nokia Logo File
NOL - Nokia Operator Logo
OTB - Nokia OTA bitmap
PAT - Gimp Pattern
PBM - Portable Bitmap
PCL - Page Control Language
PDB - Palm Pilot
PDF - Portable Document Format
PGM - Portable Greyscale
PI1 - Degas & Degas Elite
PIC - Softimage
PIC - Psion Series 3 Bitmap
PIC - Rayshade
PIC - Bio-Rad confocal
PIX - Alias Image File
PIX - PABX background
PNM - Portable Image
PRC - Picture Gear Pocket
PS - Postscript
PSD - Adobe Photoshop
QRT - Qrt Ray-Tracer
RAD - Radiance
RAW - Raw
RAWRAW - Raw
RGB - Silicon Graphics RGB
RLA - Wavefront Raster file
SCT - SciTex Continuous Tone
TGA - Truevision Targa
VST - Vista
WBMP - Wireless Bitmap (level 0)
WRL - VRML2
XBM - X11 Bitmap
XPM - X11 Pixmap
YUV - YUV 16Bits Interleaved
YUV - YUV 16Bits

Ilustración I-2: Formatos admitidos por la herramienta de software libre XnView

- Reconocer únicamente textos con símbolos contemplados en el alfabeto español o inglés (sí que se deberán reconocer números arábigos y romanos)
- La investigación se restringirá a los diagramas de clases y componentes. Más concretamente y para evitar ambigüedad debido a las distintas versiones de UML se presentan a continuación los elementos gráficos a reconocer. Estos elementos no deben estar deformados o modificados, como por ejemplo, al representarlos a mano alzada o en tres dimensiones.

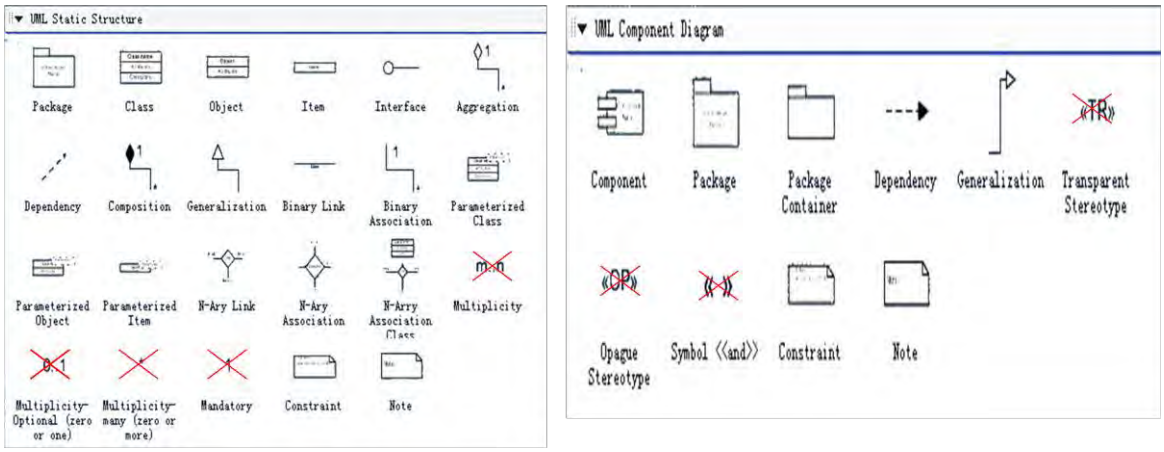


Ilustración I-3: Figuras presentes en los diagramas UML estándar OMG

Capítulo II: Estado del Arte

2.1 Gestión del conocimiento

2.1.1 **Introducción**

Diversos autores [Gómez-Pérez et al., 2004]; [Daconta et al., 2003] han señalado que la Gestión del Conocimiento engloba la adquisición y organización del mismo, utilizando usualmente para dicho fin una representación semántica. De esta forma, mediante los Sistemas de Organización del Conocimiento se construyen y gestionan los modelos conceptuales de los dominios, desde los que es posible recuperar y comprender la información contenida en los mismos.

Las diferencias entre los distintos modelos conceptuales reside en diferentes aspectos, como: el grado de capacidad semántica representada; la temática abordada; y, fundamentalmente, la finalidad que se persigue con determinada representación. Sin embargo, es posible establecer un nexo común entre los diversos esquemas de representación del conocimiento, estos son: la necesidad de un control del vocabulario y la definición previa de las relaciones semánticas que se establecerán entre los conceptos.

La definición de etapas para acometer la Gestión del Conocimiento y su ejecución organizada ha sido analizada desde diversas áreas, como: la Ingeniería del Software, las Ciencias Documentales, y la Ingeniería Ontológica. El objetivo que persigue ésta estructura en etapas es agilizar el proceso, reduciendo la inversión intelectual y favoreciendo cierto grado de consenso entre expertos.

La etapa de adquisición del conocimiento es la etapa encargada de recopilar el conocimiento que se tiene del dominio. La explicitación de ese conocimiento puede ser realizada a partir de fuentes documentales o a partir del conocimiento de los expertos. Es este uno de los principales "cuellos de botella" definidos en el proceso de Gestión del Conocimiento [Gómez-Pérez et al., 2004]. Cuando se trabaja con fuentes documentales se debe recopilar un conjunto de documentos que hayan sido seleccionados por expertos del dominio a representar. A partir de dichos documentos se debe recopilar el vocabulario y los sinónimos terminológicos presentes en dichas fuentes. Esta fase es la más laboriosa y determina la eficiencia en la generación del modelo [Antoniou y Harmelen, 2004]; [Gómez-Pérez et al., 2004]. En las siguientes etapas, y a partir de este vocabulario, se establecerán las estructuras jerárquicas y relacionales entre los conceptos, así como las restricciones que afectan a estas asociaciones.

2.1.2 **Relevancia de la gestión del conocimiento**

Son diversos los motivos implicados en la importancia de la gestión del conocimiento y por tanto responsables de la alta atención investigadora que se le dedica. Fensel [Fensel et al., 2002] destacó que la importancia de la Gestión del Conocimiento se fundamenta, principalmente, en los siguientes factores: la sobrecarga de información, la necesidad de recuperar información mediante consultas eficientes que no se limiten a las palabras clave, la falta de autoridad literaria en entornos Web y la carencia de sistemas automáticos de PLN especializados en la gestión del conocimiento. De forma análoga, Antoniou [Antoniou y Harmelen, 2004] indica que esta necesidad viene impulsada por la necesidad de las organizaciones, los colectivos y de distintas disciplinas por estructurar la información, con el fin de apoyar los procesos comunicativos, competitivos y de desarrollo interno.

En la actualidad, resulta evidente que el desarrollo de Internet ha provocado un aumento desmesurado de la información. Este mismo fenómeno puede ser observado en otras redes y organizaciones. Sin la gestión adecuada de ese conocimiento resulta imposible no sólo su recuperación sino incluso saber de su existencia. Las alertas que manualmente indicaban los empleados veteranos sobre nueva información relevante y su ubicación es claramente ineficaz dada la actual velocidad de producción de información.

Un entorno en el que está casuística es evidentemente relevante es el mundo empresarial. En él la gestión del conocimiento es vital, la utilización de asesoría virtual dirigida al

marketing, la formación de empleados y la presentación ante el mundo comercial ya no se pueden entender sin la adquisición, organización y divulgación de ese conocimiento.

2.1.3 **Conocimiento Implícito y Explícito**

Según Nonaka [Nonaka y Takeuchi, 1995] el conocimiento se puede dividir en dos categorías: conocimiento explicito e implícito. El conocimiento explicito, es aquel que es precisado de forma estructurada y documentada. Por otro lado, aquel conocimiento que es fruto de experiencias y percepciones se denomina conocimiento implícito o tácito.

Las organizaciones gestionan fácilmente el conocimiento explícito a partir de la información tecnológica, sin embargo, la gestión del conocimiento implícito presenta mayores dificultades por residir éste casi exclusivamente en las mentes humanas [Duffy, 2000].

La mayor parte del conocimiento implícito que se genera en un proyecto se debe a las interacciones entre su personal. Con el fin de poder reutilizar este conocimiento en futuros proyectos las empresas recurren a rotar su personal entre los diferentes proyectos o al almacenamiento de experiencias mediante informes, minutas, videos y/o grabaciones.

La explicitación del conocimiento es por tanto necesario para utilizar el conocimiento en el futuro. Lamentablemente, en la práctica, la extensa documentación producida a lo largo del ciclo de vida de un proyecto suele disuadir de su consulta a los nuevos equipos de trabajo [Conklin, 2001]. Por otra parte, la inestabilidad de las plantillas contribuye a que la mayor parte del conocimiento generado en un proyecto se pierda [Weiser y Morrison, 1998].

2.1.4 **Representación del conocimiento**

La gestión del conocimiento comprende diversos aspectos como la identificación de conceptos, la creación, distribución o representación del mismo. Diversos autores han señalado la importancia de la representación y la visualización de la información en la gestión del conocimiento [Hoetzlein, 2007]. En concreto, Dolk [Dolk, 1984] señalaba que la gestión tenía dos grandes componentes la gestión del conocimiento y la representación de la misma. A continuación se analiza la representación del conocimiento en la Ingeniería del Software.

2.1.4.1 Representación del conocimiento en Ingeniería del Software

La representación del conocimiento en Ingeniería del Software consiste en representar de manera formalizada conceptos y componentes software para formar una estructura útil para su posterior gestión, implementación, recuperación y reutilización. Los elementos que intervienen en la representación del software son generados a lo largo del ciclo de vida del proyecto de software. Partiendo de la base de que para recuperar es necesario que previamente se encuentre el recurso, es conveniente que: exista un sistema de recuperación adecuado y que el repositorio sobre el que se hace la búsqueda tenga recursos documentales suficientes [Llorens et al., 2004].

Habitualmente se ha considerado que los únicos documentos asociados a la representación del software debían ser documentos tecnológicos generados durante el ciclo de vida del software como por ejemplo los que contienen especificaciones de requisitos, definición de casos de uso, modelos de clases y estados, especificación de las secuencias o codificación. Sin embargo, existen otros tipos de documentación menos relacionados con el proceso de ingeniería pero de gran trascendencia en el producto software [Ramesh y Jarke, 2001]. Así García [García et al., 2008] defienden la incorporación de este tipo de documentos en los corpus y aseguran que la relevancia de los elementos presentes en estos documentos debe estar a la par con la de los demás elementos asociados a los proyectos software. Ejemplos de estos documentos serían minutas, informes, resúmenes, actas de reuniones, o incluso emails entre los implicados en la concepción y desarrollo de productos se pueden tener en cuenta para completar la representación. Una ventaja de este tipo de documentación es que suele estar más cercana a las experiencias y percepciones frutos de las interacciones entre el personal implicado en los proyectos. Es decir, recoge parte del conocimiento tácito imposible de capturar en documentos más tecnológicos (ver epígrafe 2.1.3).

2.1.4.2 Grado de automatización sobre la tipología documental

La diferente tipología documental afecta al grado en que los distintos documentos son incorporados a un Sistema de Representación de la Información. Esto se debe a la necesidad de procesar automáticamente los documentos en favor de agilizar la representación del conocimiento o de la información contenida en ellos. De esta forma, frecuentemente se descartan documentos por la naturaleza compleja y problemática de su procesamiento automático. Un ejemplo de esta casuística podría ser aquellos documentos

que necesitan la utilización de un OCR para capturar el texto y un corrector ortográfico posterior, ya que ambas etapas son sensibles a la producción de errores [Velasco, 1998].

En concreto, los documentos se pueden dividir en dos grandes grupos en función del grado de automatización en la captura de su semántica:

1. En el primer bloque están incluidos aquellos documentos cuya carga semántica esencial dependa de texto en lenguaje natural. Estos documentos pueden tener información asociada mediante etiquetas, por las zonas ocupadas por el texto en el documento o por cualquier otro método de fácil interpretación por un programa informático. Ejemplo de este tipo de documentos son: los documentos de texto libre, los documentos con plantillas y los documentos con etiquetas siempre cuando su formato no sea un impedimento para su tratamiento. Ejemplos en la Ingeniería del Software se corresponden con documento de requisitos, juegos de pruebas, correos electrónicos, etc.

2. El segundo grupo de documentos está formado por documentación difícilmente interpretable por una máquina bien por su formato o por su forma de representar la semántica. Ejemplos en la Ingeniería del Software son: videoconferencias entre personal del proyecto (tanto por el audio como el video) y los diagramas UML.

En particular, en este segundo grupo de documentos, los diagramas son documentos muy relevantes semánticamente y bastante frecuentes en los proyectos. Su semántica viene dada por la combinación de texto en lenguaje natural y estructuras gráficas. Estas estructuras gráficas tienen asociada una información semántica, accesible a la interpretación humana, que depende del tipo de diagrama.

Para ilustrar el valor de estos diagramas, baste el ejemplo de los diagramas de clases y los diagramas de objetos de UML [Rumbaugh et al., 1998]. Estos diagramas son capaces de representar conceptos por medio de clases y atributos de esas clases, las relaciones entre clases, los objetos (instancias de las clases) y los métodos (funciones) asociados a las clases. Asimismo, pueden también soportar restricciones por el tipo de datos, el rango de valores, la multiplicidad entre clases y las restricciones entre relaciones.

2.1.4.3 Representación de Información mediante modelos en Ingeniería del Software

Los modelos entidad-relación y los modelos UML[1] (Unified Modeling Language) se utilizan para representar el conocimiento en la Ingeniería del Software. Son modelos concebidos para diseñar aplicaciones informáticas y se utilizan principalmente en el desarrollo conceptual y la planificación de los desarrollos propios de la Ingeniería del Software.

Los modelos relacionales y los modelos de entidad relación [Chen, 1976] son modelos de representación de datos estructurados que responden a la necesidad de almacenar información en bases de datos relacionales. El modelo Entidad-Relación utiliza un lenguaje de modelado conceptual y es utilizado en la definición de esquemas conceptuales para las bases de datos relacionales. Ambos modelos emplean diferentes grados de abstracción. El modelado conceptual representa entidades, incluyendo sus atributos y las relaciones existentes entre esas entidades, restringiéndolas por su cardinalidad.

UML está más enfocado a la representación de elementos en la programación orientada a objetos. El estándar UML [Rumbaugh et al., 1998] establece un lenguaje y un esquema de representación común que permiten visualizar, especificar, construir, documentar y comunicar todo tipo de artefactos[2].

UML define varios modelos para la representación de los sistemas. Estos modelos son vistos y manipulados por los usuarios por medio de vistas gráficas que pueden construirse a partir de los modelos básicos. UML define los siguientes tipos de diagramas:

- Estructurales: Visualizan, especifican, construyen y documentan los aspectos estáticos de un sistema.
    - Diagramas de Clases: Presenta un conjunto de clases, interfaces, colaboraciones y las relaciones entre ellas;

[1] UML http://www.uml.org/

[2] Un artefacto, en el área de la Ingeniería del Software y basándose en UML, es una pieza tangible de información (un modelo, un elemento de un modelo o un documento) [Jacobson et al., 1999].

- Diagramas de Objetos: Representa un conjunto de objetos y sus relaciones. Se utilizan para describir estructuras de datos, instantáneas de las instancias de los elementos encontrados en los diagramas de clases;
- Diagramas de Componentes: Muestra un conjunto de componentes y sus relaciones. Describen la vista de implementación estática de un sistema;
- Diagramas de Despliegue: Muestra un conjunto de nodos y sus relaciones. Se utilizan para describir la vista de despliegue estática de una arquitectura.

- De Comportamiento: Visualizan, especifican, construyen y documentan los aspectos dinámicos de un sistema.
  - Diagramas de Secuencia: Es un diagrama de interacción que resalta la ordenación temporal de los mensajes.
  - Diagramas de Colaboración: Es un diagrama de interacción que resalta la organización estructural de los objetos que envían y reciben mensajes.
  - Diagramas de Estados-Transiciones: Representa una máquina de estados constituida por estados, transiciones, eventos y actividades.
  - Diagramas de Actividades: muestra el flujo de actividades de un sistema.
  - Diagramas de Casos de Uso: Representa un conjunto de casos de uso y actores y sus relaciones

2.2 Reutilización en la Ingeniería del Software

La reutilización es un concepto que siempre ha estado presente en cualquier actividad dirigida a la resolución de problemas. En definitiva se trata de agrupar los problemas por su forma de resolverlos. A partir de las soluciones de cada grupo de problemas se generaliza un método de resolución. Así, cuando un nuevo problema tiene que ser resuelto aprovechamos el esfuerzo empleado en la resolución de otros problemas similares al ya tratado.

La reutilización en la Ingeniería del Software presenta mayores dificultades que en otras disciplinas. El motivo se debe a la imposibilidad de realizar desarrollos informáticos mediante la unión directa de componentes software [Llorens, 1996], pues la funcionalidad de la combinación resultante difícilmente cumplirá la especificación requerida. Otro problema radica en el hecho de que le Ingeniería del Software no está tan asentada como

otras disciplinas, por lo que es difícil establecer progresos en la reutilización en una actividad que está en permanente cambio.

A pesar de estos inconvenientes la reutilización del software sigue pareciendo una de las formas más prometedoras para optimizar el desarrollo de los proyectos software.

La unidad de reutilización son los componentes software definidos por Neighbors [Neighbors, 1984] como: "Aquellos elementos que especifican la semántica de un cierto dominio de aplicación". Si tenemos en cuenta que la reutilización se puede aplicar a cualquier etapa del ciclo de vida del software (análisis, diseño, codificación, pruebas, mantenimiento y nuevos desarrollos) [Neighbors, 1984], [Ning, 1993], [Jarzabek, 1993] se debe extender esta definición y considerar que los componentes software pueden ser, además de código ejecutable, cualquier tipo de información útil para el desarrollo de nuevos proyectos. Así, el concepto de componentes tiene diferentes matices según el autor consultado, una definición dada por Bass en el 2000 [Bass et al., 2000, p. 10] define componente como "Una implementación software de alguna funcionalidad. La reutilización sin cambios se puede producir en distintas aplicaciones".

Esta definición de componente software permite estructurar la reutilización en tres niveles:

1. Reutilización de especificaciones
2. Reutilización de diseño
3. Reutilización de código

Las ventajas obtenidas para la reutilización serán mayores cuanto más alto sea el nivel de abstracción de la etapa en que se aplique. Por ejemplo, la reutilización de código es la menos productiva por ser posterior a las otras dos etapas y requerir menor esfuerzo [Velasco, 1998].

Según Llorens [Llorens y Velasco, 1995] la reutilización del software implicada la existencia de un repositorio organizado de componentes software junto con un sistema automático que permita su gestión. En base a estos repositorios se debe de poder localizar analizar y determinar la calidad de los componentes software así como poder establecer las relaciones existentes entre ellos.

Entre las propiedades deseables que debe tener los repositorios destacan [Llorens, 1996]:

- Capacidad de almacenamiento y recuperación de componentes

- Posibilidad de creación, edición, visualización y ensamblado de componentes por parte del usuario.
- Disponibilidad de un formalismo de representación de componentes estándar.
- Facilidad de incorporación de nuevos componentes.
- Navegabilidad a partir de esquemas organizativos

En definitiva, el repositorio es la estructura en la que se apoya la reutilización del software. Una vez aceptada la necesidad de crear y mantener repositorios cabe plantearse si la inversión en costes reduce sensiblemente los beneficios de la reutilización. En este sentido las investigaciones se centran en encontrar técnicas automáticas que permitan la construcción de repositorios de calidad. Una tendencia para abordar la reutilización del software es el análisis de dominios. Mediante la definición de los dominios es posible que la reutilización debido a su mayor estructuración conceptual.

2.2.1 **Análisis de Dominios**

Neighbors [Neighbors, 1981] definió el análisis de dominios como: "the activity of identifying the objects and operations of a class of similar systems in a particular problem domain". Además, resaltó la importancia de dirigir el análisis de dominios hacia la reutilización del análisis y el diseño en lugar de orientarlo únicamente hacia la reutilización del código [Neighbors, 1981], [Neighbors, 1984].

El primer desarrollo de análisis de dominios fue el proyecto DRACO [Neighbors, 1984]. En este proyecto se demostró que la adquisición y estructuración del conocimiento de un dominio permitían la generación semiautomática de aplicaciones software a partir de la reutilización de componentes software.

El primer modelo para realizar análisis de dominios fue propuesto por Rubén Prieto [Prieto-Díaz, 1987]. El autor define el análisis de dominios como: "Un proceso por el que la información utilizada en el desarrollo de sistemas de software se identifica, se captura y se organiza con el propósito de hacerla reutilizable cuando se crean sistemas nuevos". Además, enfatiza la necesidad de realizar análisis de dominios y la opción de capturar conocimiento de cualquier tipo de fuente [Prieto-Díaz, 1990]. Por último, propone utilizar tesauros facetados para organizar los dominios en reutilización [Prieto-Díaz, 1991].

Posteriormente se han propuesto más modelos y métodos dirigidos al análisis de dominios con el objetivo de automatizar al máximo la reutilización del software.

Otras aproximaciones llevadas a cabo por algunos investigadores han propuesto la utilización de las leyes bibliométricas para automatizar etapas del análisis de dominios en la reutilización de software [Godin et al., 1995]. En 1996, Lloréns [Llorens, 1996], propone de forma práctica un tesauro de software para simplificar la reutilización de software.

Una aplicación de estos enfoques puede encontrarse en la herramienta CASE de análisis de dominios. Desarrollada por Frakes, Prieto y Fox [Frakes et al., 1998], incluía tesauros y contemplaba como entrada cualquier tipo de recurso generado en el proyecto.

Posteriormente, Velasco [Velasco, 1998] propone explorar técnicas documentales, estadísticas o de inteligencia artificial para generar dominios, tipo tesauro, automáticamente. Ante la falta de eficacia de los métodos de forma individual se propone combinarlos entre sí. Por otro lado, Irene Díaz [Díaz, 2001] propone un repositorio denominado RSHP para representar cualquier objeto UML ya que un sistema debe de ser capaz de recuperar cualquier objeto del desarrollo del proyecto de software. En su tesis además se propone basar la generación automática en técnicas lingüísticas y no estadísticas (mediante un repertorio manual de términos bajo determinado patrón léxico, de manera que cada patrón debía ser capaz de mapear una relación semántica de UML).

Por último, hay que recalcar que diversos estudios constatan que el éxito de un proyecto de reutilización de software está relacionado con el análisis de dominios y la calidad del repositorio creado [Morisio et al., 2002], [Menzies y Justin, 2003].

2.3 Algoritmos de Visión Artificial

2.3.1 **Introducción**

Actualmente, existen multitud de técnicas para el procesado de imágenes. La mayoría de las aplicaciones incorporan mecanismos y procedimientos para modificar las imágenes de acuerdo a ciertos parámetros, y así poder analizar la imagen desde distintos puntos de vista.

Para el desarrollo de esta tesis el estudio se ha centrado en las técnicas básicas para el reconocimiento de objetos y formas como la umbralización y la detección de bordes. También ha sido necesario aplicar técnicas de modificación de las imágenes, como remuestreos, para poder mejorar la captación de elementos.

Estas técnicas se apoyan en funcionalidades elementales dadas por cualquier software especializado, como son los filtros (por color, por tonalidad, binarios…), pixelaciones, esqueletización, generación de histogramas, etc.

Por otro lado, se han descrito aplicaciones para el procesamiento y la detección de figuras en las imágenes. Este sector está muy desarrollado y se pueden encontrar aplicaciones muy interesantes, como es el caso de VisionLab [Mitov, 2008], que permite la detección de figuras en imágenes e incluso de movimiento en videos.

Por lo general estas aplicaciones van destinadas a tareas muy específicas como el reconocimiento facial pero incluyen opciones más básicas que pueden ser integradas en el desarrollo de otro tipo de trabajos.

También, hay que resaltar que existen herramientas relacionadas con el reconocimiento de formas aún más especializadas, cuya cometido es la lectura del texto presente en las imágenes. Estas aplicaciones reciben la denominación de OCR (Optical character recognition).

En definitiva, las técnicas de visión artificial utilizadas en ese trabajo son:

- Umbralización
- Detección de bordes
- Remuestreo
- Detección de líneas
- Detección de formas por circularidad
- Reconocimiento óptico de caracteres

2.3.2 **Umbralización**

La umbralización consiste en encontrar un valor diferencial que discrimine dentro de una imagen entre los píxeles que representan el fondo y los que representan los objetos. Por lo general, este valor diferencial se obtiene a partir del histograma[3] de la imagen que contiene la información de los pixeles en su tonalidad de gris [Vernon, 1991]. Esta técnica resulta tanto más precisa cuanto más homogénea sea la imagen y mayor sea la diferencia de tonalidad entre el fondo y los píxeles que se contienen información útil.

[3] El histograma de una imagen representa la frecuencia relativa de los niveles de gris de la imagen.

Los experimentos muestran que los peores resultados se obtienen en imágenes que presentan diferentes puntos e intensidades de iluminación y en las que abundan sombras y reflejos.

2.3.3 **Detección de bordes**

Un borde es el límite entre dos regiones de una imagen, es decir, entre dos zonas con diferentes niveles de gris.

Si se fija cualquier trayectoria en el recorrido de los píxeles de una imagen, e interpretando sus valores de nivel de gris como pertenecientes a una función, los bordes se corresponderán con los saltos bruscos de dicha función. Aplicando, por tanto, el cálculo de derivadas podremos detectar los bordes.

La primera derivada de un punto de una imagen se obtiene utilizando el módulo del gradiente [Robert, 1965] [Danielsson, 1990] de ese punto. La segunda derivada, se obtiene de forma análoga utilizando el Laplaciano [Huertas, 1986].

2.3.3.1 Derivada de primer orden

Los operadores de gradiente ortogonal detectan bordes horizontales y verticales .Están definidos por las ecuaciones [1] y [2]:

[1] $$\frac{\partial f(x,y)}{\partial x} = \Delta_x = \frac{f(x+d_x,y) - f(x,y)}{dx} \qquad \Delta_x = f(i+1,j) - f(i,j)$$

[2] $$\frac{\partial f(x,y)}{\partial y} = \Delta_y = \frac{f(x,d_y+y) - f(x,y)}{dy} \qquad \Delta_x = f(i,j+1) - f(i,j)$$

Las ecuaciones se implementan en forma digital haciendo la convolución de una imagen con una región o máscara de 3x3.

Los operadores de Prewitt, Sobel, Roberts y Frei-Chen son operadores de dos etapas, una para la detección de bordes horizontales y otra para los bordes verticales.

En la Ilustración II-1 se muestran algunas máscaras de convolución (se distingue con H o V horizontal de vertical).

	Roberts			Prewitt			Sobel			Frei-Chen		
H	0	0	-1	1	0	-1	1	0	-1	1	0	-1
	0	1	0	1	0	-1	2	0	-2	$\sqrt{2}$	0	$-\sqrt{2}$
	0	0	0	1	0	-1	1	0	-1	1	0	-1
V	-1	0	0	-1	-1	-1	-1	-2	-1	-1	$-\sqrt{2}$	-1
	0	1	0	0	0	0	0	0	0	0	0	0
	0	0	0	1	1	1	1	2	1	1	$\sqrt{2}$	1

Ilustración II-1: Mascaras de convolución (horizontales y verticales)

2.3.3.2 Derivada de segundo orden

El Laplaciano es un operador escalar de segunda derivada para funciones de dos dimensiones definido por la ecuación [3]:

[3] $$\nabla^2 f(x,y) = \frac{\partial^2}{\partial x^2} f(x,y) + \frac{\partial^2}{\partial y^2} f(x,y)$$

La ecuación se implementa en forma digital haciendo la convolución de una imagen con una región o máscara de 3x3.

El principio para la definición de este operador es que el coeficiente asociado al píxel central de la máscara sea positivo y los coeficientes asociados a los píxeles exteriores sean negativos o nulos, de tal forma que la suma de todos los coeficientes sea cero.

Al hacer la convolución para cada punto de la imagen con cualquiera de las máscaras de la Ilustración II-2 la respuesta es cero siempre que el punto central tenga el mismo valor que sus vecinos. Se muestran en la siguiente ilustración algunas mascaras del operador Laplaciano.

1	-2	1	-1	-1	-1	0	-1	0
-2	4	-2	-1	8	-1	-1	4	-1
1	-2	1	-1	-1	-1	0	-1	0
Laplaciano 1			Laplaciano 2			Laplaciano 3		

Ilustración II-2: Mascaras de convolución (Laplaciano)

El Laplaciano cuenta con algunas desventajas con respecto a los operadores de primer orden:

- El Laplaciano es más sensible al ruido en imágenes
- Genera bordes dobles
- No existe información direccional de los ejes detectados

Al ser este operador sensible al ruido, se utiliza después de binarizar la imagen para mejorar el resultado.

2.3.4 **Remuestreo**

Se llama remuestro o interpolación a la técnicas que permiten magnificar una imagen de dimensiones *N x M* para obtener otra de dimensiones *pN x pM* [Escalera, 2001].

2.3.4.1 Remuestreo por aproximación:

Cada píxel de la nueva imagen toma el valor de color de su píxel correspondiente en la imagen original. Al ser varios los píxeles que en la imagen aumentada se asocian a cada píxel de la original, se generan bordes más definidos y por tanto más propensos a mostrar pixelamiento. Es por tanto, el método más rápido y a la vez el que genera imágenes de menor calidad.

2.3.4.2 Remuestreo bilineal:

Se basa en promediar los valores de color más cercanos (vecindad de 2 x2) para obtener el color del pixel resultante. Mejora la calidad respecto de la técnica anterior al producir bordes más suaves.

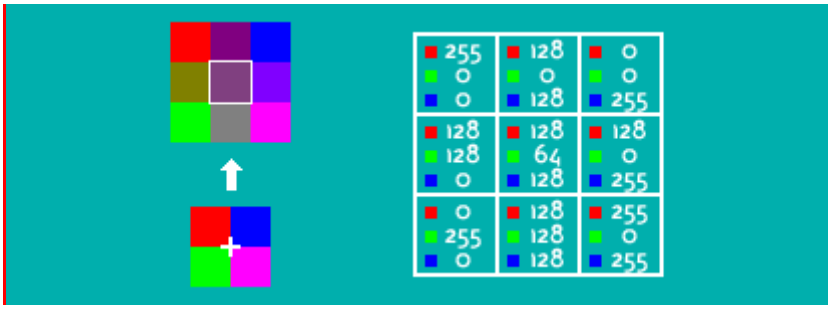


Ilustración II-3: Ejemplo de remuestreo por aproximación[4]

2.3.4.3 Remuestreo bicúbico:

Esta técnica sigue la línea de consultar los valores de color de los vecinos. Las: diferencias respecto a la técnica anterior se deben a la consideración de una mayor vecindad de píxeles (4 x 4), y a la ponderación de los valores de color de los mismos atendiendo a su proximidad. Es la técnica más utilizada con fines profesionales a pesar de su mayor coste computacional.

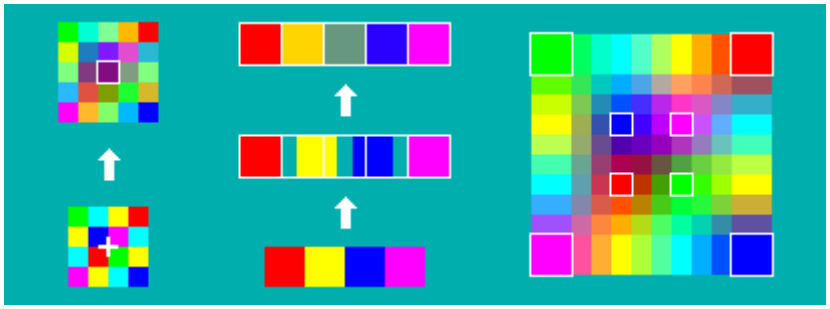


Ilustración II-4: Ejemplo de remuestreo bicúbico[5]

2.3.5 **Detección de Líneas Rectas**

Una de las formas que con mayor frecuencia interesa detectar en una imagen es la línea recta. Su detección, al igual que para otras formas, se realiza a menudo a partir de los píxeles candidatos a bordes. En la mayoría de los casos las imágenes de trabajo presentan ruido.

---

[4] Imagen tomada de https://sites.google.com/a/artsgrafiques.org/jmayo-bcn/mp-02-tractament-dimatge-en-mapa-de-bits/uf-2-2-obtencio-d-imatges-digitals

[5] Imagen tomada de la misma web de la nota anterior.

Entre los métodos de detección de rectas en imágenes destacan:

1. Transformada de Radon
2. Transformada de Hough
3. Método de RANSAC
4. Método de acumulación

2.3.5.1 Transformada de Radon

La transformada de Radon [Radon, 1917], aplicada a la detección de rectas en imágenes, parte del siguiente principio:

El valor de la suma a lo largo de una trayectoria recta puede decidir sobre la existencia o no de rectas sobre dicha trayectoria.

[4] $$R_f(p,\varphi) = \int_{x\cos\varphi + y\sin\varphi = p} f(x,y)dxdy$$

Los valores más altos de la integral corresponden con las líneas rectas de la imagen.

La complejidad del método es $O(ND^2)$ [Voss et al., 2004] con N como el número de los puntos y D como el número medio de las parejas discretas de valores (p,φ).

Las líneas rectas precisas se pueden obtener por el método de mínimos cuadrados o por optimización lineal [Voss y Suesse, 2001].

2.3.5.2 Transformada de Hough

Hough [Hough, 1962] interpreto, a diferencia de Radon, la ecuación [5] como una ecuación de parámetros (x,y) y variables (p,φ).

[5] $$p = x\cos\varphi + y\sin\varphi$$

Este enfoque permite extender la transformada de Radon a otras figuras como los círculos y reducir la complejidad que con este método es $O(ND)$ para la detección de rectas y $O(ND^2)$ para la detección de círculos [Voss et al., 2004]. Siendo N el número de puntos y D el de ángulos.

2.3.5.3 Método de RANSAC (random simple consensus)

El método de RANSAC [Fischler y Bolles, 1981] basa su teoría en la búsqueda de puntos sobre la vecindad de una línea. Pero a diferencia del método de Radon las líneas se buscan aleatoriamente seleccionando dos puntos al azar.

La selección de los dos puntos determina una recta candidata que se valida tras contar todos los puntos de su vecindario y comprobar que se supera un umbral.

Si no se combina este criterio con otros como la densidad de puntos en la recta se suelen obtener errores en la detección de rectas.

La complejidad de este método es $O(N^2)$ [Voss et al., 2004]

2.3.5.4 Método de acumulación

Este método de acumulación [Voss et al., 2004] combina las ventajas del método de Radon y del método RANSAC y su complejidad es $O(N^2)$ si se investigan al azar N parejas de puntos y $O(N^3)$ si se estudian exhaustivamente la $N(N-1)$ parejas de puntos posibles.

Inspirado en la transformada de Hough se presentó el método de acumulación como una generalización. Se consigue mejorar el coste computacional conservando las buenas propiedades de la transformada de Hough como son: la resistencia al ruido, la posibilidad de detectar formas ocluidas (incluso las que no tienen posibilidad de enlazado de bordes) y la facilidad para representar una figura con un borde de un píxel de grosor.

2.3.6 **Detección de formas por circularidad**

La circularidad [González, 1999] es una característica adimensional e invariante a la orientación, capaz de distinguir formas. Se calcula a partir del área y perímetro de la forma:

[6] $$circularidad = \frac{perímetro^2}{área}$$

Siendo el área el número de píxeles del interior del contorno, y el perímetro la longitud determinada por los píxeles del borde.

Se ha utilizado, por ejemplo, para la preclasificación de las señales de tráfico en triangulares, circulares o cuadradas [Moreno et al., 2006].

Área	l^2	$\sqrt{3}\, l^2/4$	πr^2
Perímetro	$4l$	$3l$	$2\pi r$
Perímetro ^{2}/ Área	16	20.8	12.56

Ilustración II-5: Valores de la circularidad para las señales de tráfico según su forma.

2.3.7 **Optical character recognition (OCR)**

El software OCR, extrae el texto contenido en una imagen en un formato compatible con los editores de texto. Los OCR, además de reconocer texto en varios idiomas pueden identificar su estilo y formato e incluso trabajar con textos manuscritos y partituras musicales.

Sin embargo, la efectividad de los OCR es aproximadamente del 90% siempre que las imágenes sean de alta calidad. Las resoluciones idóneas de las imágenes están en torno a los 300 ppp y los textos deben estar preferiblemente limpios sin cuadros de texto, encolumnados y demás elementos de diseño.

Existen diversos paquetes de software (libre y comercial) cuya funcionalidad es el reconocimiento de texto en imágenes.

2.3.7.1 Yunmai OCR SDK

Yunmai OCR SDK (http://www.ocrsdks.com) tiene entre sus funcionalidades el reconocimiento de matriculas de automóviles, tarjetas de visita y documentos en general. Su ámbito de aplicación son empresas dedicadas a las telecomunicaciones y las finanzas; y se puede integrar en los procesos de producción de dispositivos como escáneres, cámaras o los smartphones.

Es software propietario de la empresa, con sede en China, Yunmai Technology (fundada en 2002) aunque es posible su uso gratuito los 30 días de duración del periodo de pruebas.

2.3.7.2 Tesseract

Tesseract (https://code.google.com/p/support/) es una librería desarrollada C y C++ por la compañía Hewlett Packard a partir de 1985 durante diez años. En 2005, una vez liberado como código abierto, pasa su distribución a Google bajo la licencia Apache.

Una de sus principales ventajas es la posibilidad de adaptarse, mediante entrenamiento, a nuevos idiomas. Por defecto incluye la posibilidad de reconocer textos en español, inglés, francés, alemán, portugués, italiano y holandés.

2.3.7.3 SmartScore X2

SmartScore X2 (http://www.musitek.com) es un software OCR propietario desarrollado por la empresa Musitek Corporation. Originalmente, en 1991, se distribuyo para Windows como MIDISCAN. En 1998 pasó a denominarse SmartScore, distribuyéndose una versión nueva para Windows 98, y otra al año siguiente para Macintosh Power PC.

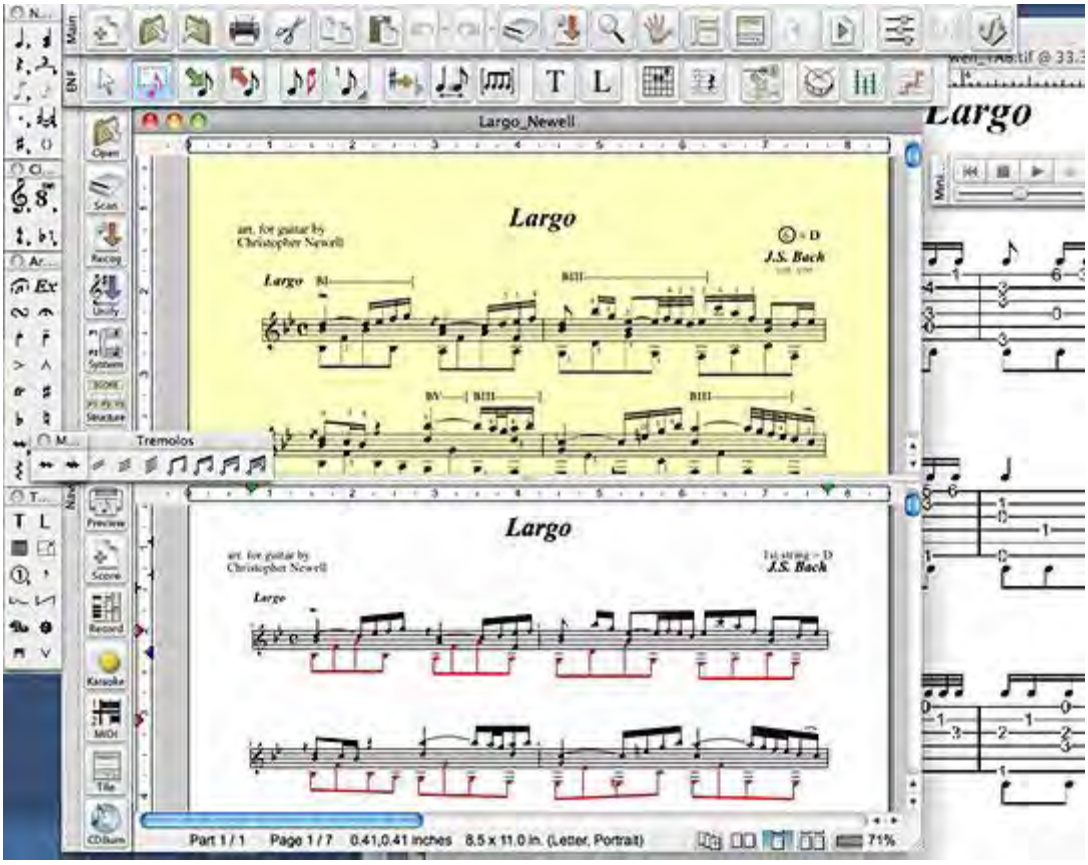


Ilustración II-6 Captura SmartScore X2

SmartScore tiene como funcionalidad el reconocimiento óptico de caracteres sobre partituras de música que pueden reproducirse como archivos en formato MIDI.

2.3.7.4 SimpleOCR

SimpleOCR (http://www.simpleocr.com) es un software especializado en escanear documentos de texto. Fue desarrollado por la empresa SimpleSoftware (fundada en 2002). En la actualidad es software libre.

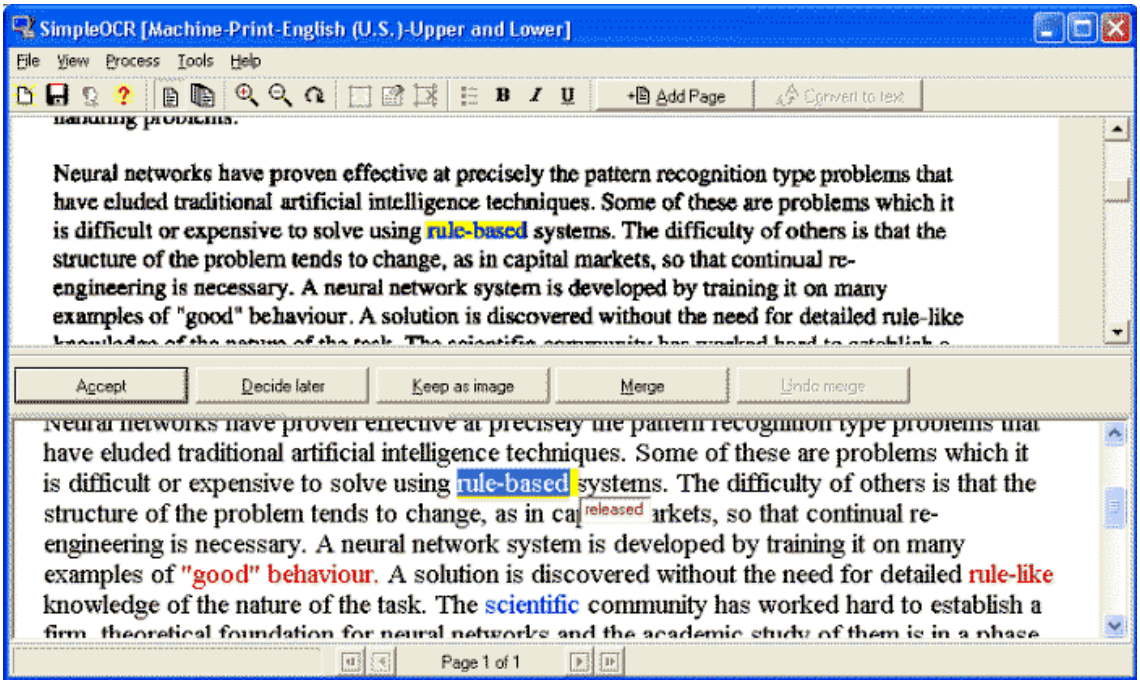


Ilustración II-7 Captura pantalla SimpleOCR

2.3.7.5 Screenworm

Screenworm (http://www.screenworm.de/) es un software OCR de escritorio desarrollado para la plataforma Mac OS X. Es propiedad de la empresa Funchip. Entre sus características principales destacan el reconocimiento de texto y su traducción, la captura de imágenes y la capacidad de reproducir textos.

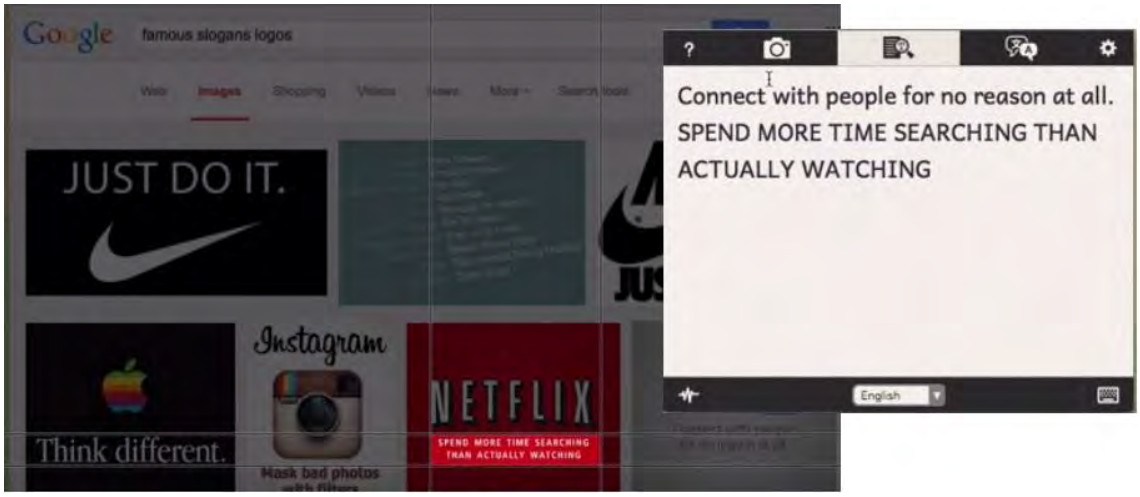


Ilustración II-8 Captura Screenworm

2.3.7.6 Scantron

Scantron (http://www.scantron.com). Es un software de escritorio perteneciente a la empresa Scantron Corporation. Además de realizar reconocimiento óptico de caracteres, puede reconocer automáticamente repuestas de test (encuestas o exámenes) mediante OMR (reconocimiento óptico de marcas).

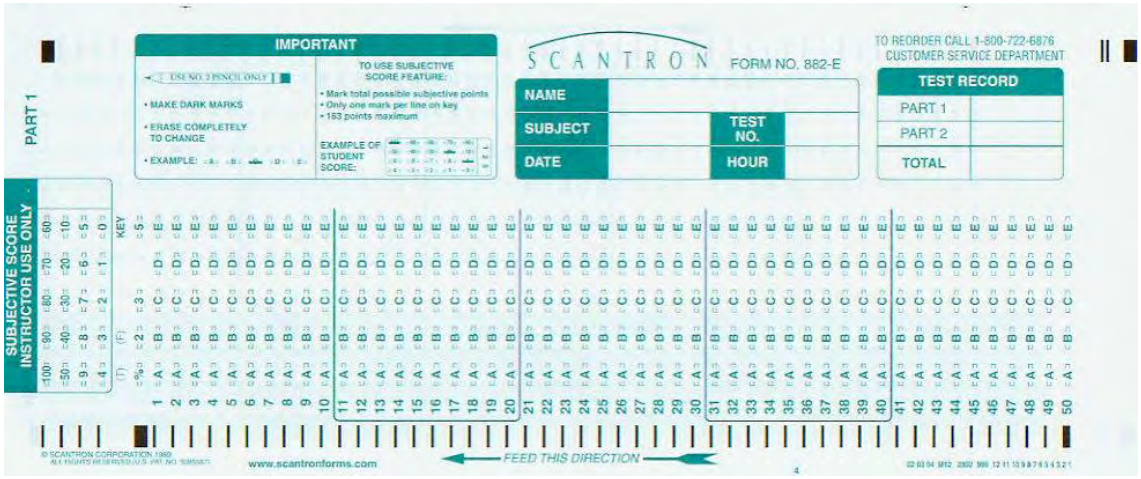


Ilustración II-9 Ejemplo de Scantron

2.3.7.7 ReadSoft

ReadSoft (http://www.lexmark.com) es un software propietario que actualmente pertenece a la empresa Lexmark. Se puede aplicar online, en su propia web, y está especializado en procesado de facturas y otras soluciones empresariales.

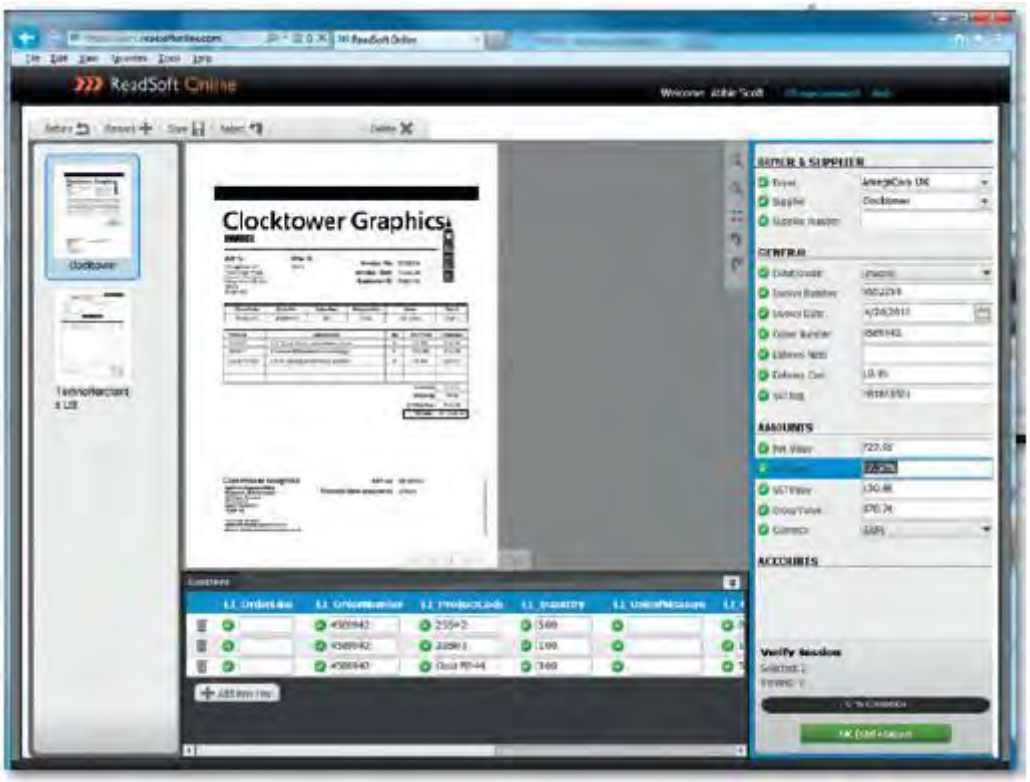


Ilustración II-10 Captura ReadSoft Online

2.3.7.8 Puma.NET

Puma.net (http://pumanet.codeplex.com) es un software con licencia BSD (Berkeley Software Distribution) desarrollado por Maxim Saplin. Tiene capacidad para reconocer 27 lenguajes diferentes en casi la totalidad de las fuentes, detectando los tamaños y los estilos de las mismas. Permite además la entrada de imágenes en diferentes formatos.

Es utilizado, como librería, por el motor de reconocimiento de Cognitive Technologies CuneiFrom, el cual permite dotar con funcionalidades OCR al Framework.Net a partir de la versión 2.0.

2.3.7.9 OmniPage

OmniPage (http://www.nuance.com/for-individuals/by-product/omnipage/index.htm) fue de los primeros software OCR distribuidos para PC (finales de los 80). Fue vendido por Caere Corporation y actualmente pertenece a Nuance Commnications. Puede reconocer un amplio abanico de idiomas, más de 120.

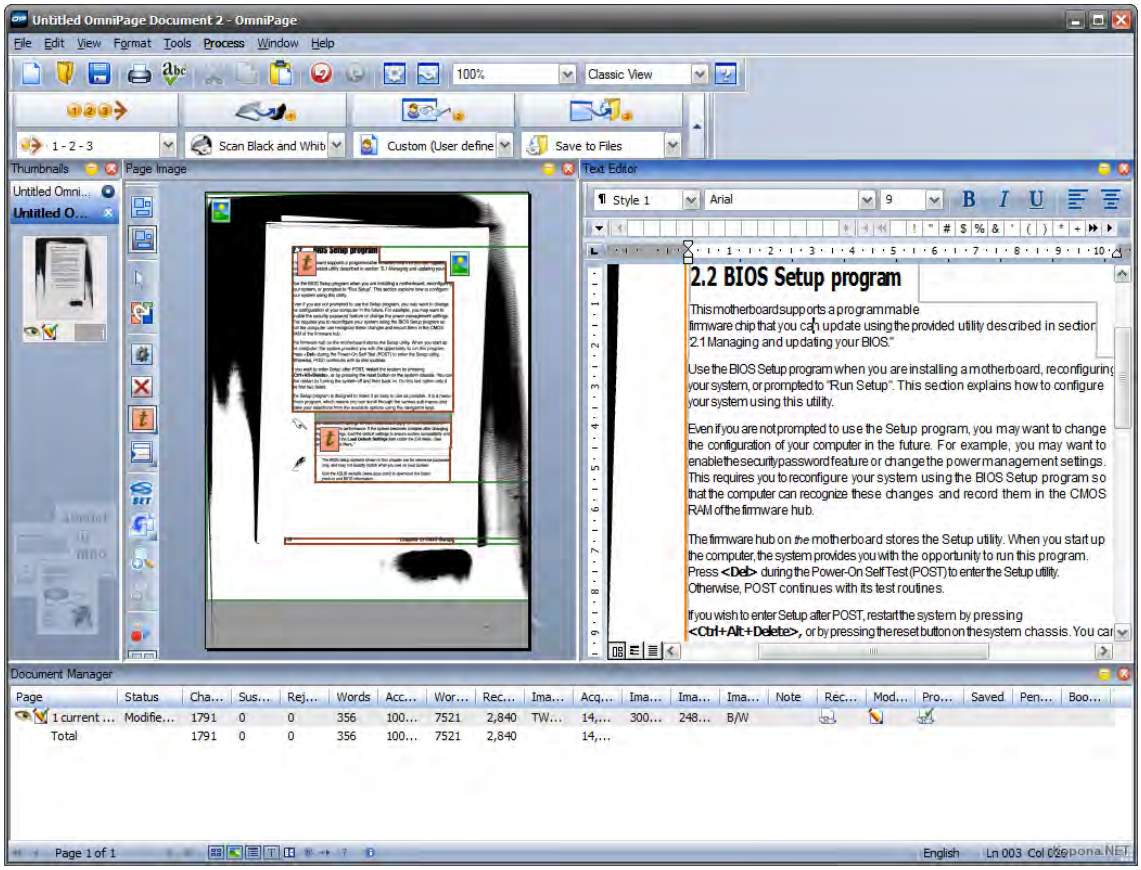


Ilustración II-11 Interfaz de OmniPage

2.3.7.10 OCRopus

OCRopus (https://github.com/tmbdev/ocropy) es un software de escritorio fruto de un proyecto liderado por Thomas Breuel y que fue desarrollado por el Centro de Investigación Alemana para la Inteligencia Articial (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) para las plataformas Unix, con distribuciones Unix y Mac OS X.

Esta herramienta ofrece OCR y análisis de documentos gratuitos. En la actualidad la patrocina Google.

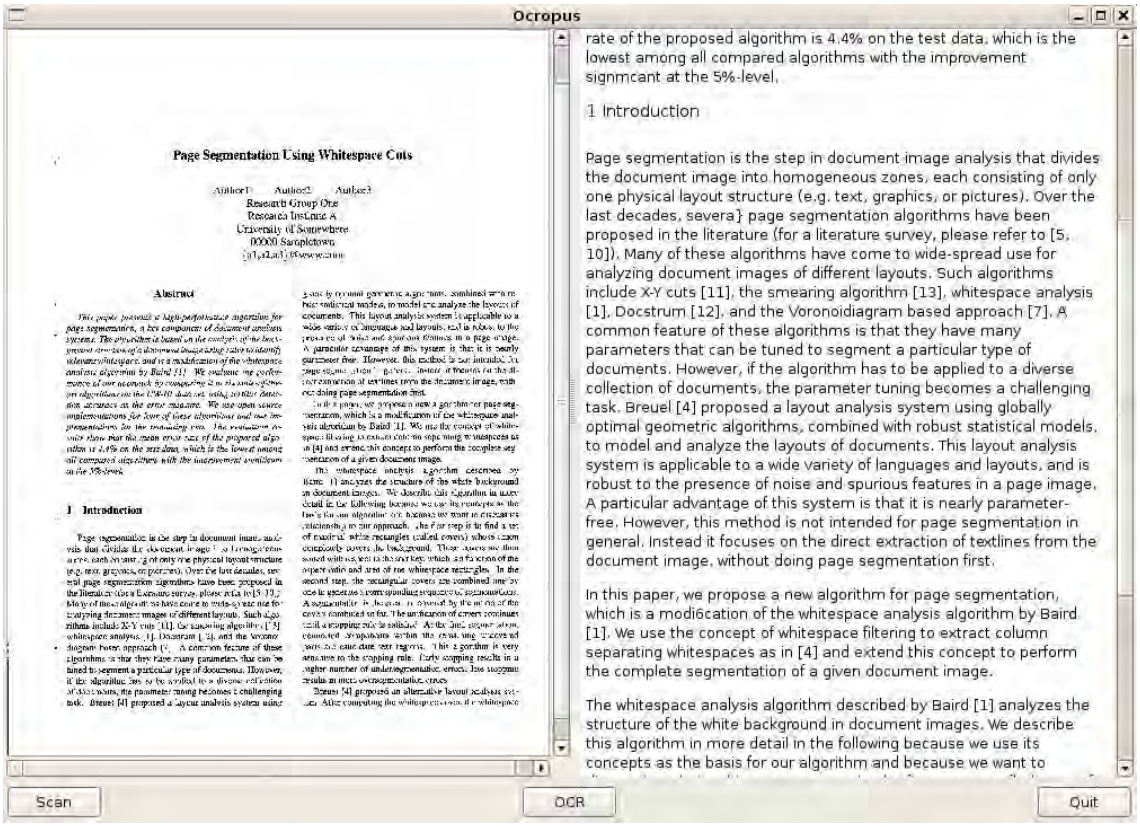


Ilustración II-12 Interfaz de OCRopus

2.3.7.11 OCRFeeder

OCRFeeder (https://wiki.gnome.org/Apps/OCRFeeder) es un software para escritorio, con licencia GPL (General Public License), desarrollado en España por Joaquim Rocha (Igalia). La aplicación utiliza un motor OCR desarrollado para GNOME.

Admite entrada de datos en formato imagen y en PDF. La posibilidad de manejo mediante una interfaz amigable o por línea de comandos le confiere una mayor usabilidad.

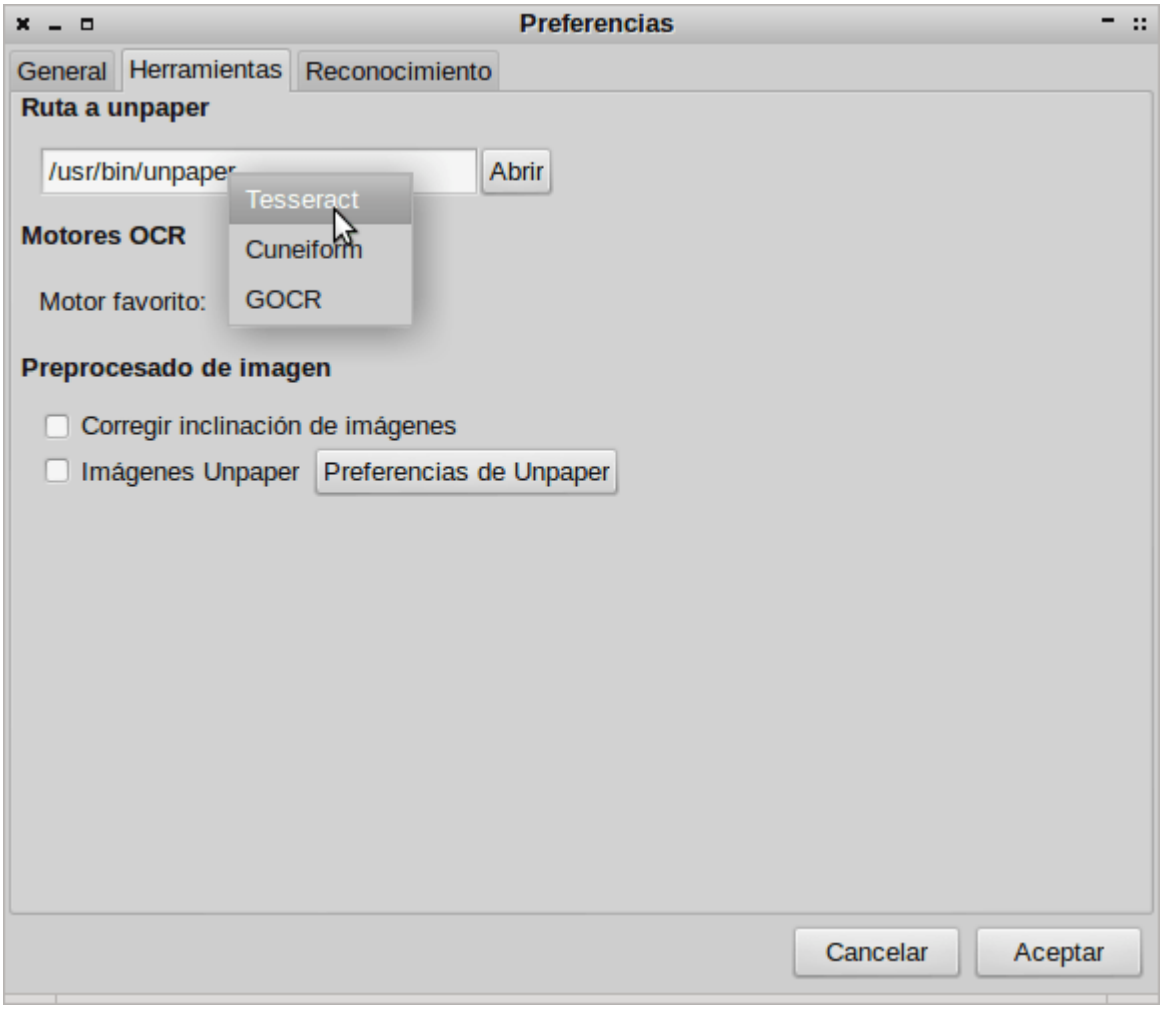


Ilustración II-13 Interfaz gráfica OCRFeeder

2.3.7.12 Ocrad

Ocrad (https://www.gnu.org/software/ocrad/) es un software OCR de escritorio con licencia GPL (General Public License). Se desarrollo en España, dentro del proyecto GNU, por Antonio Díaz Díaz. El software de separar, en páginas impresas, las columnas o bloques de texto. El formato admitido en las imágenes de entrada es Netpbm y el del texto resultante UTF-8.

Ilustración II-14 Ejemplo ejecución Ocrad

2.3.7.13 Nicomsoft OCR SDK

Nicomsoft OCR SDK (http://www.softpedia.com/get/Programming/SDK-DDK/Nicomsoft-OCR-SDK.shtml) es un software propietario, con motor OCR, que pertenece a la empresa Nicomsoft. Se apoya en algoritmia matemática propia de campos como la visión por computador y la inteligencia artificial.

2.3.7.14 Microsoft Office OneNote 2007

Microsoft Office OneNote 2007(https://www.onenote.com). Es un software con licencia propietaria, perteneciente a la compañía Microsoft, desarrollado para multisistema con funcionalidad OCR en sus últimas versiones.

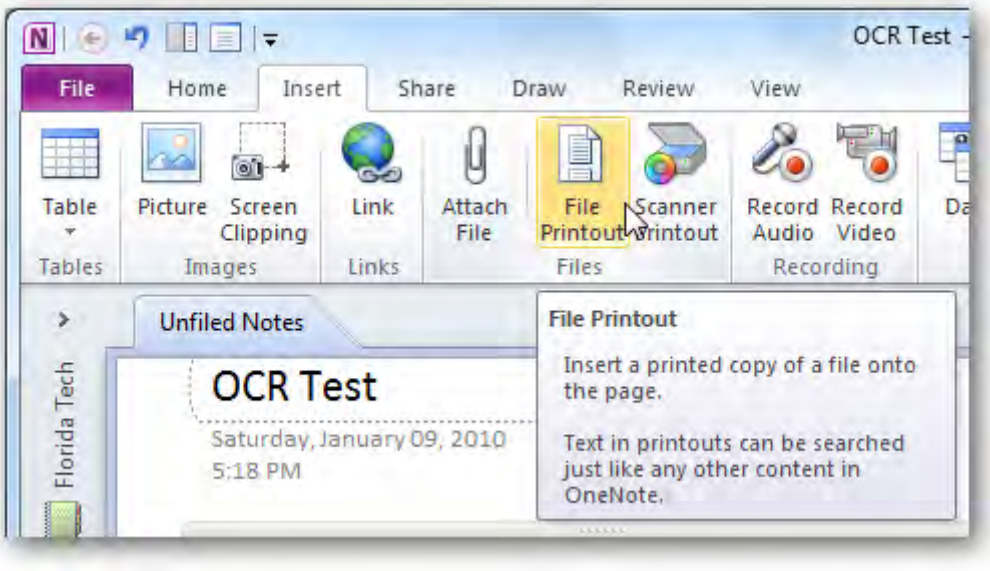


Ilustración II-15 Interfaz Onenote 2007

2.3.7.15 Microsoft Office Document Imaging

Microsoft MODI (https://support.microsoft.com/es-es/kb/982760) es un software, al igual que OneNote, con licencia propietaria de Microsoft. Sus funcionalidades como herramienta OCR le permiten editar y hacer anotaciones mediante metadatos en documentos escaneados. Admite archivos en formato TIFF. Fue distribuido por primera vez con el Sistema Operativo Windows XP dentro del paquete ofimático Office.

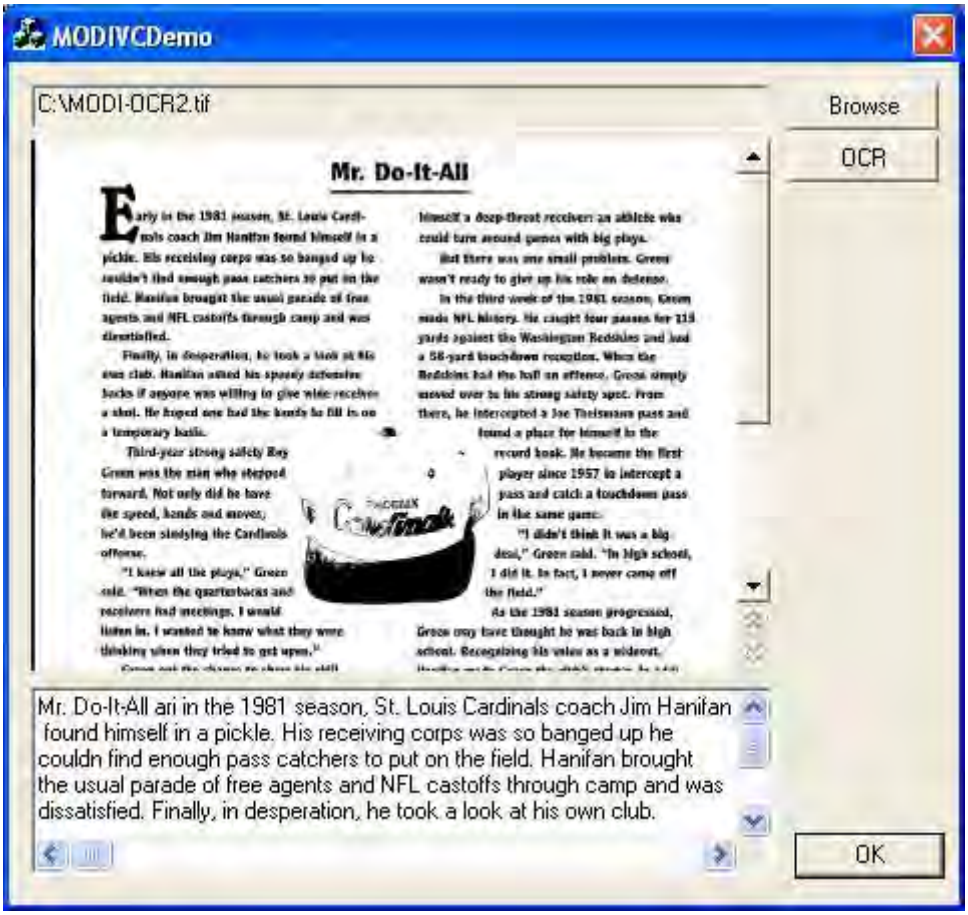


Ilustración II-16 Interfaz MODI

2.3.7.16 MeOCR

MeOCR (http://www.meocr.com) es un software de escritorio libre, propiedad de la compañía MeOCR. Incorpora un motor OCR preparado para reconocimiento de texto durante el escaneado o posteriormente tras cargar los ficheros. Admite, en más de 15 idiomas, varios tipos de formatos y, varias escalas de tonos de grises y colores. Además, es compatible con Microsoft Word.

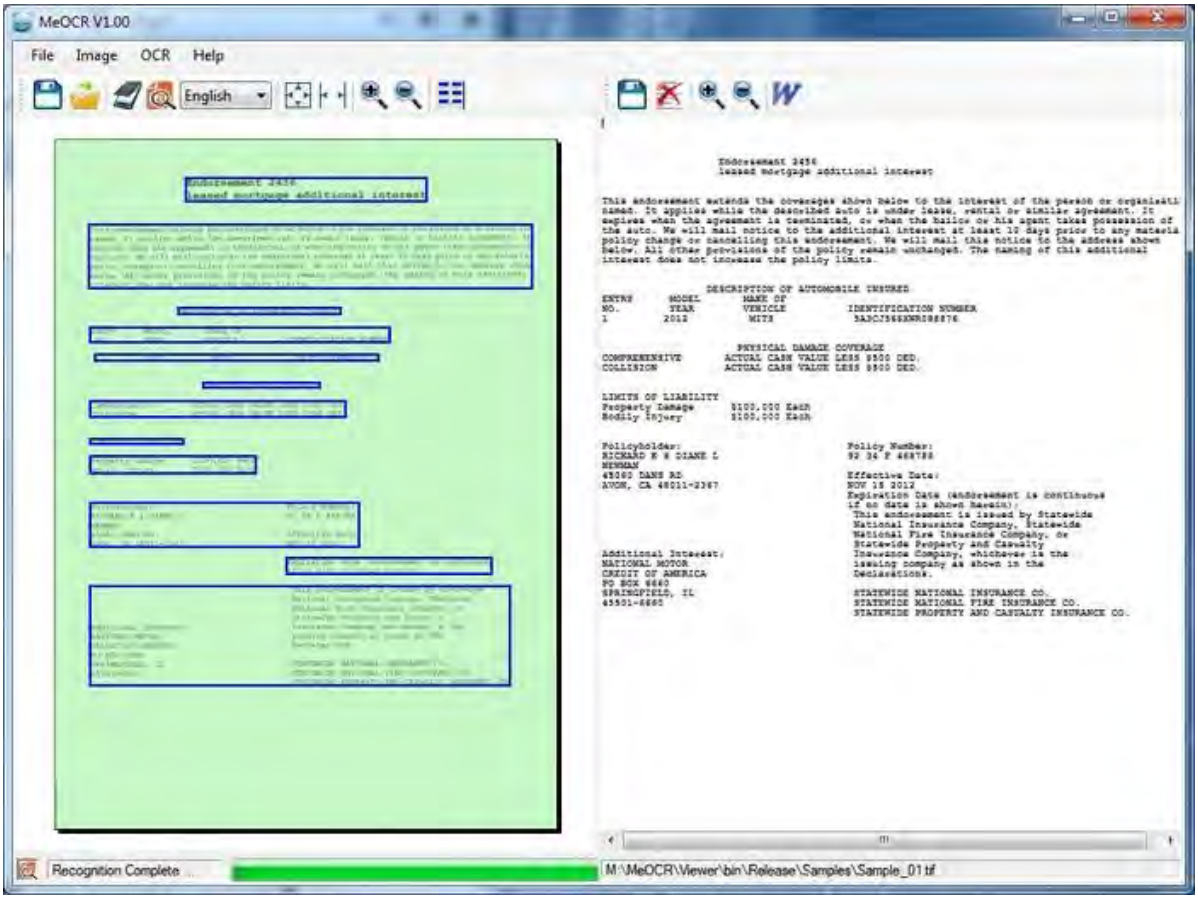


Ilustración II-17 Interfaz de MeOCR

2.3.7.17 MathOCR

MathOCR (http://sourceforge.net/projects/mathocr/) es software libre que incorpora un motor OCR cuya finalidad es la identificación de expresiones matemáticas admitiendo diversos idiomas.

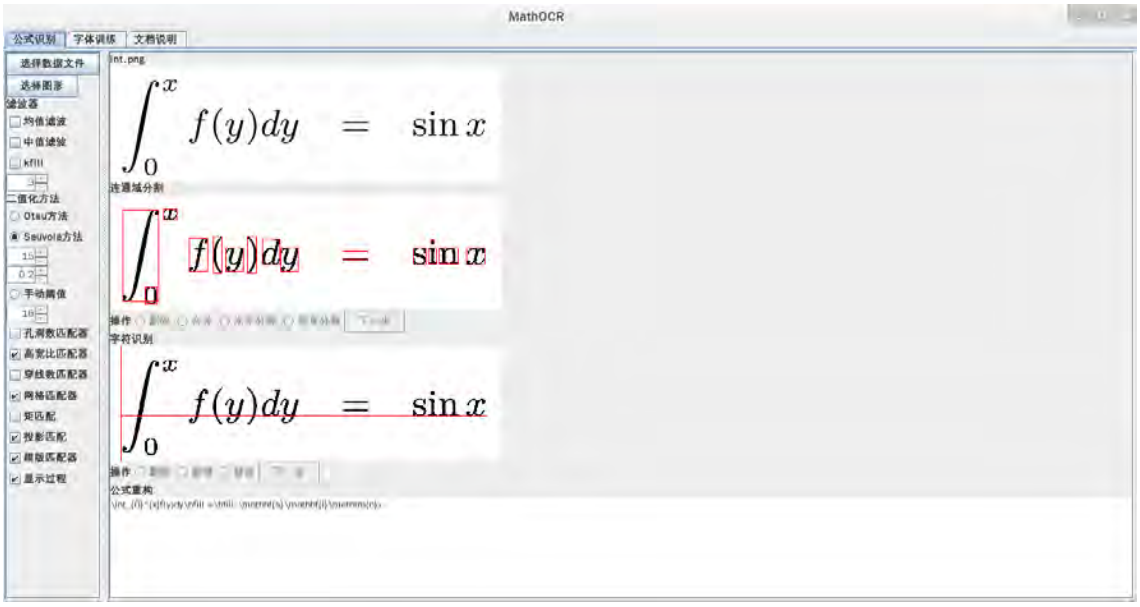


Ilustración II-18 Interfaz de MathOCR

2.3.7.18 LEADTOOLS

LEADTOOLS (https://www.leadtools.com) es un software con licencia propietaria perteneciente a la empresa LEAD Technologies. Esta aplicación incorpora un software OCR en una amalgama de herramientas (de dibujo vectorial, multimedia y médicas,) que permite incluir a sus aplicaciones técnicas digitales.

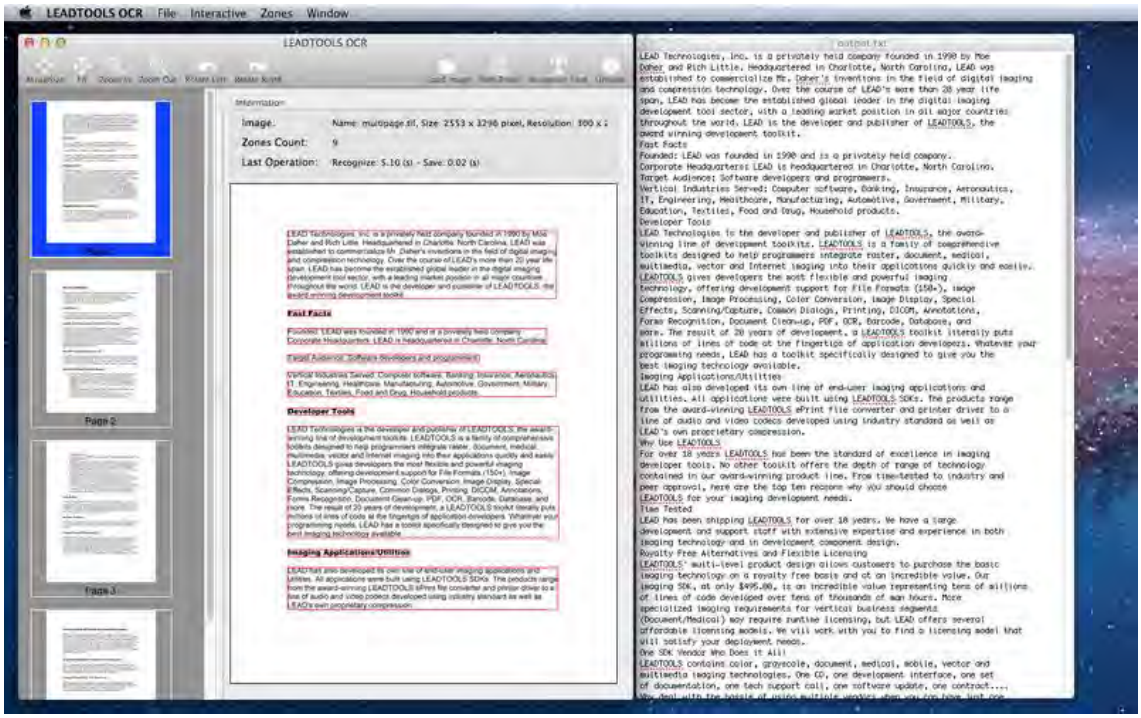


Ilustración II-19 Interfaz de LeadTools

2.3.7.19 GOCR

GOCR (http://jocr.sourceforge.net) es un software con licencia GPL (General Public License) desarrollado por Jörg Schulenburg. Está capacitado transformar imágenes con formato Netpbm. Aunque se le achacan problemas para algunos tipos de tipografías.

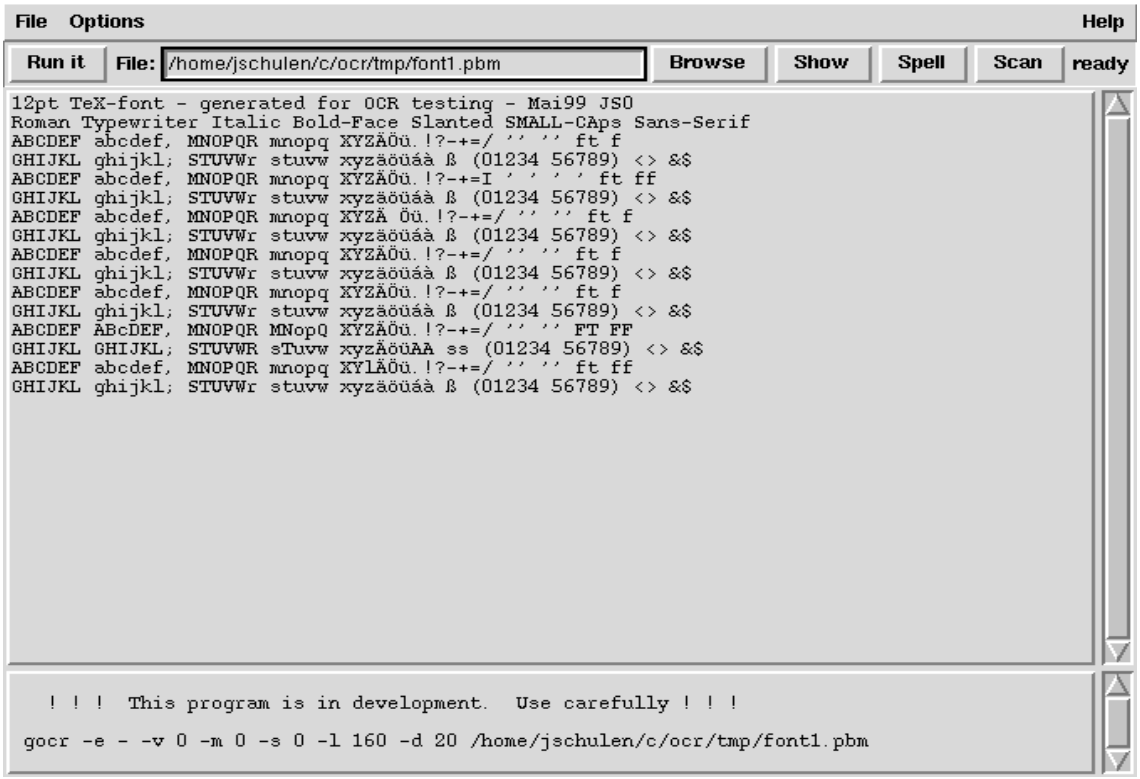


Ilustración II-20 Interfaz GOCR

2.3.7.20 FreeOCR

FREEOCR (http://www.free-ocr.com/es.html) es una herramienta libre online con motor OCR apto para el reconocimiento en diversos idiomas. Admite imágenes en los formatos BMP, TIFF, JPG, GIF. También puede procesar la primera página de los documentos que estén en formato PDF.

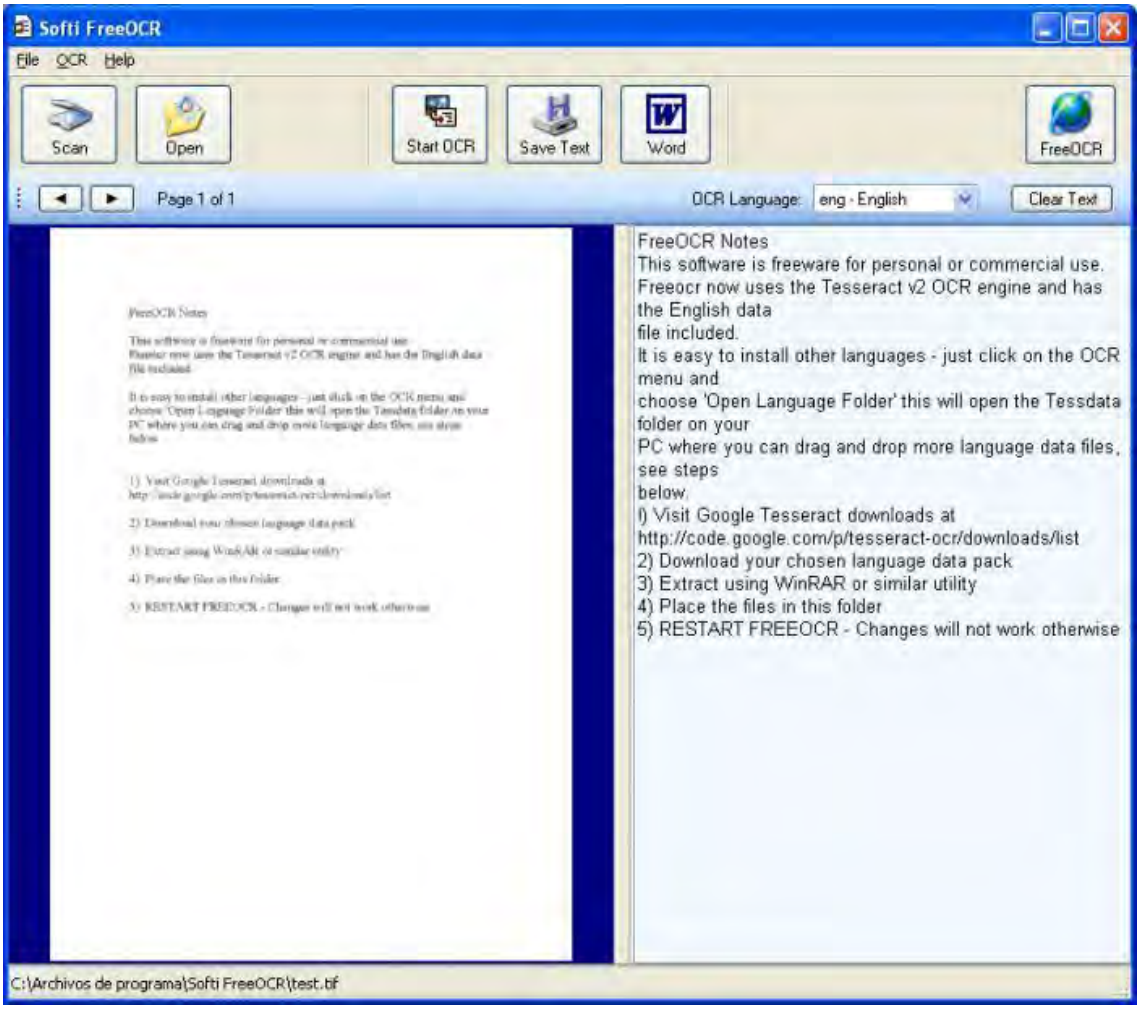


Ilustración II-21 Interfaz de FreeOCR

2.3.7.21 ExperVision TypeReader & RTK

Expervision (http://www.expervision.com/ocr-software) es un software con licencia propietaria, perteneciente a la empresa Exper-OCR que incorpora un motor OCR para el

procesado de formularios. Es capaz de reconocer los textos presentes en multitud de soportes: libros, capturas de pantalla, cheques, tarjetas, cartas bancarias, etc….

Ilustración II-22 Interfaz ExperVision

2.3.7.22 Dynamsoft OCR SDK

Dynamsoft OCR SDK (http://www.dynamsoft.com) es software, con licencia privada, desarrollado por la compañía Dynamsoft. Además de herramientas OCR proporciona un lector de código de barras.

2.3.7.23 CuneiForm

Cueniform (http://en.openocr.org) es una herramienta software, con de licencia BSD (Berkeley Software Distribution), desarrollado por Cognitive Technologies en 1996. Su motor OCR se basa en las tipologías de las letras y no requiere aprendizaje por

reconocimiento de patrones. Incorpora diversos alfabetos lo que le permite el reconocimiento de múltiples idiomas. A partir 2007 el software quedó liberado.

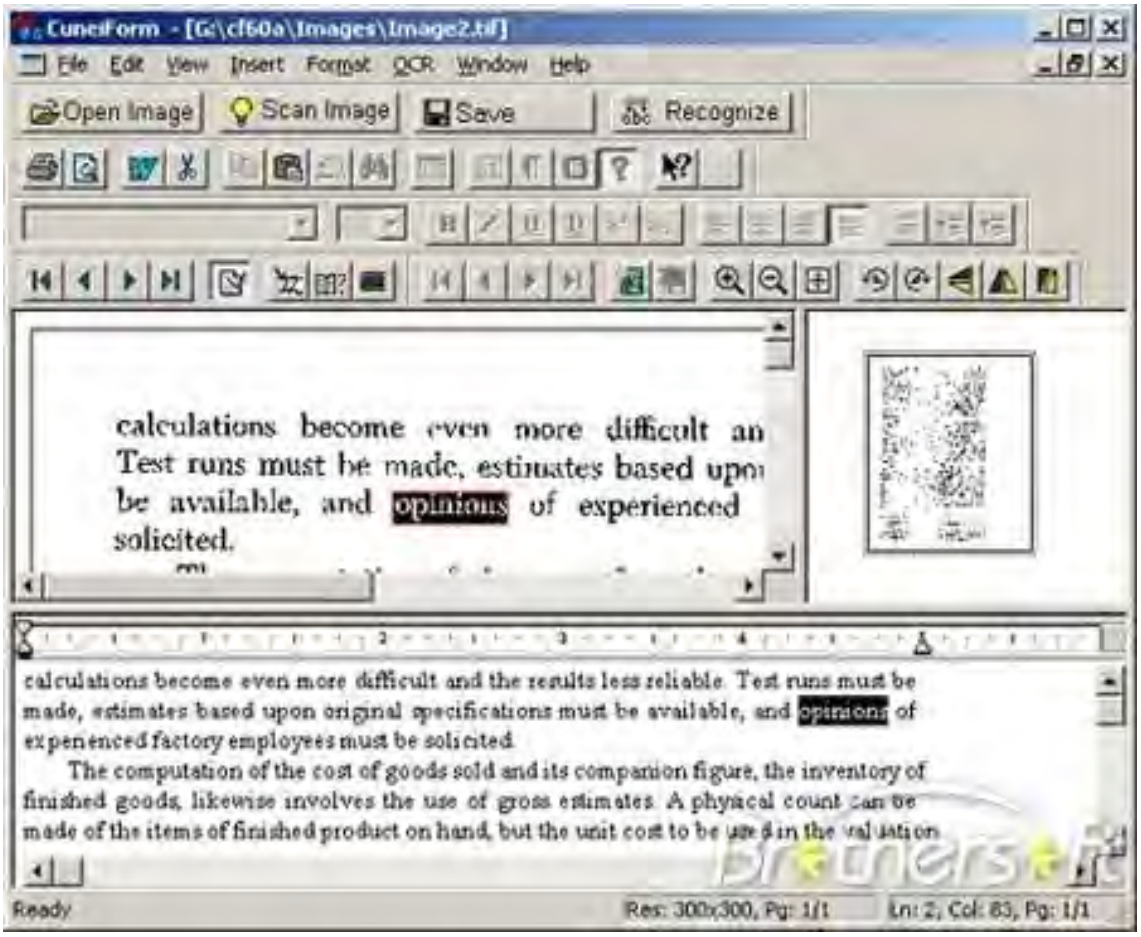


Ilustración II-23 Interfaz de CuneiForm

2.3.7.24 Asprise OCR SDK

Asprise OCR SDK (http://asprise.com/royalty-free-library/ocr-api-for-java-csharp-vb.net.html) es un software con licencia propietaria creado por la empresa Asprise. Su motor de reconocimiento de caracteres le permite identificar también códigos de barras.

2.3.7.25 AnyDoc Software

AnyDoc Software (https://www.onbase.com/en/product/onbase-anydoc) fundada con la denominación de Microsystems Techonology Inc en 1989, es una empresa que explota un software de desarrollo propio, denominado ECM, para la captura de información textual presente en imágenes o documentos escaneados.

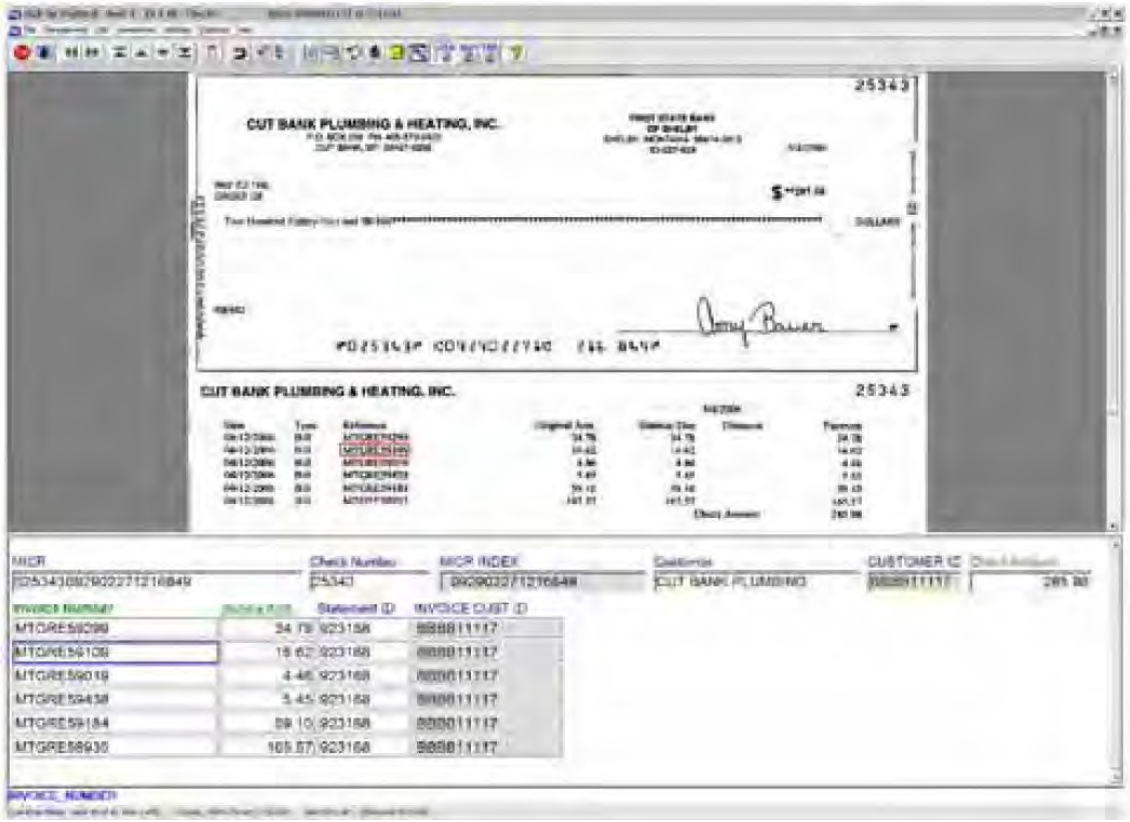


Ilustración II-24 Interfaz de AnyDoc Software

2.3.7.26 AliusDoc AD-SCI

AliusDoc (http://www.aliusdoc.com) es una empresa con desarrollos enfocados en el mundo de la medicina. Su software de reconocimiento textual está bajo licencia propietaria.

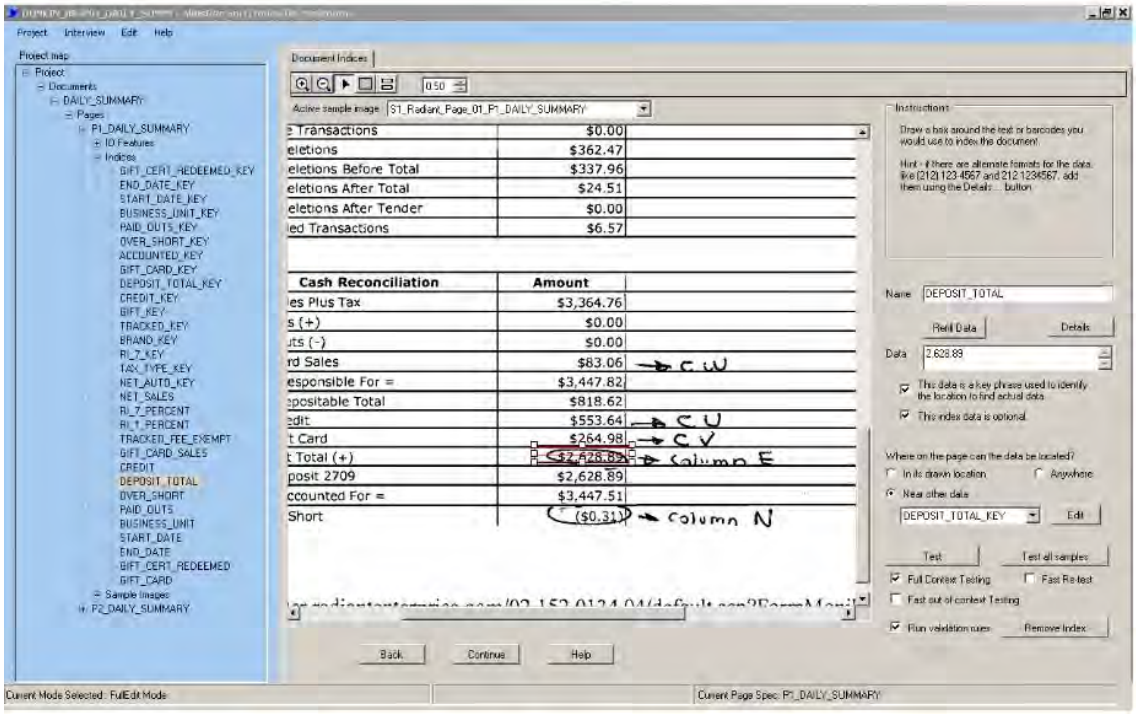


Ilustración II-25 Interfaz AliusDoc

2.3.7.27 ABBYY FineReader

Abbyy FineReader (http://www.abbyyeu.com/es/) es un software con licencia propietaria perteneciente a la empresa ABBYY. Esta compañía proporciona soluciones empresariales para reconocimiento textual en dispositivos PC y smartphones. Permite traducir los textos extraídos mediante consultas a diccionarios en tiempo real. Incorpora un lector de tarjetas de visitas. Entre los formatos admisibles están los habituales de Microsoft Office (docx, xls, ppt, etc…), los pdf, html, csv, txt, epub, fb2.

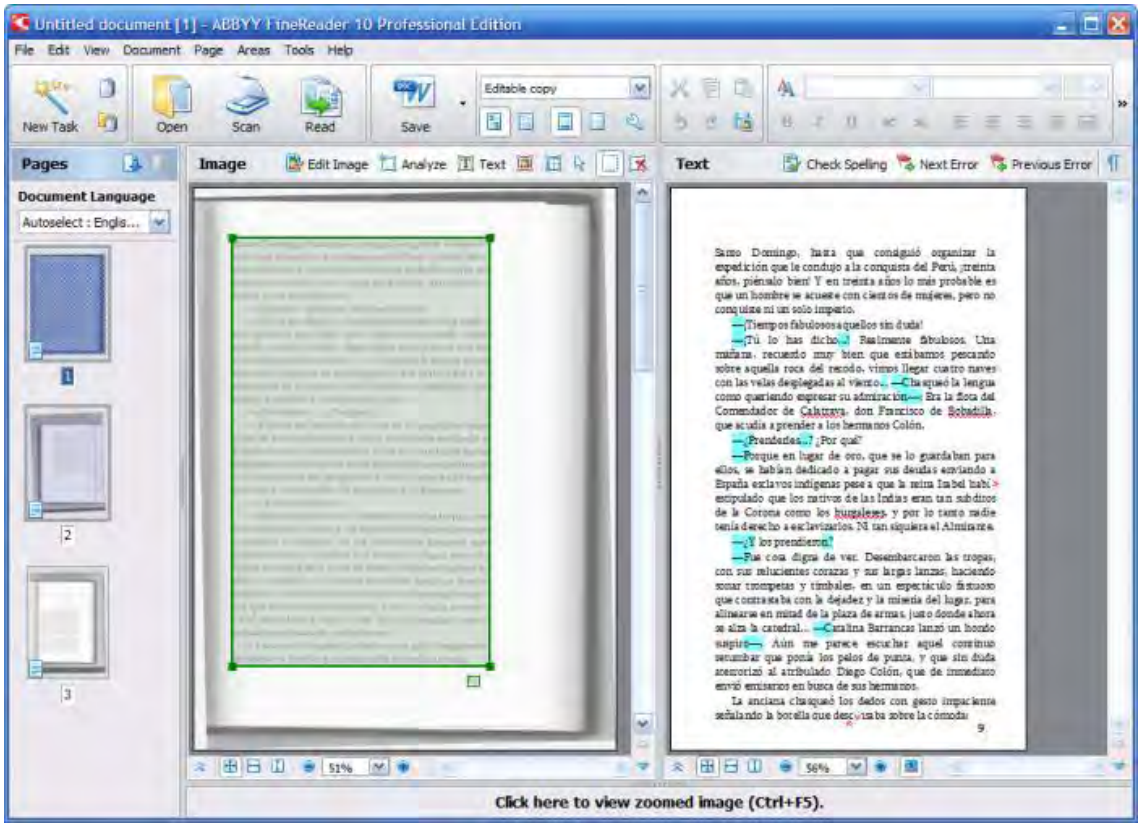


Ilustración II-26 Interfaz de ABBYY FineReader

2.3.7.28 (a9t9)FreeOCR

(a9t9) Free OCR (http://blog.a9t9.com/p/free-ocr-software.html) es una herramienta de uso libre que proporciona un motor OCR capaz de procesar imágenes desenfocadas, textos de escritura manual y diseños artísticos con porcentajes de acierto razonables en el reconocimiento de textos.

Ilustración II-27 Interfaz (a9t9) FreeOCR

2.3.7.29 Resumen Herramientas OCR

A continuación se muestra una tabla con el resumen de las herramientas y sus características.

Nombre	Última versión estable	Año de publicación	Licencia	Online	Windows	Mac OS X	Linux	BSD	Lenguaje de programación	SDK?	Lenguajes	Fuentes	Formatos de entrada
Yunmai OCR SDK	1.0	2013	Propietario	Sí	Sí	Sí	Sí	Sí	Java, C++, C, object pascal, objective-C	Sí	14	Cualquier fuente impresa	TXT, PDF
Tesseract	3.04	2015	Apache	No	Sí	Sí	Sí	Sí	C++, C	Sí	36	?	Text, hOCR, otros con diferentes interfaces de usuario o API
SmartScore	?	?	Propietario	No	Sí	Sí	No	No	?	?	?	?	
SimpleOCR	3.5	2008	Propietario	No	Sí	No	No	No	?	?	?	?	
Screenworm	1.0	2014	Propietario	No	No	Sí	No	No	Objective-C++	No	57	?	TXT
Scantron	?	?	Propietario	No	Sí	No	No	No	?	?	?	?	
ReadSoft	?	?	Propietario	No	Sí	No	No	No	?	?	?	?	
Puma.NET	?	?	BSD	No	Sí	No	No	No	C#	Sí	28	Cualquier fuente impresa	
OmniPage	19	2013	Propietario	Sí	Sí	Sí	Sí	No	C/C++, C#	Sí	125	Fuentes de máquinas y manuscritas	DOC/DOCX, XLS/XLSX, PPTX, RTF, PDF, PDF/A, Searchable PDF, HTML, Text, XML, ePUB, MP3
OCRopus	0.6	2012	Apache	No	No	No	Sí	No	Python	?	?	?	hOCR, HTML, TXT
OCRFeeder	0.7.11	2009	GPL	No	No	No	Sí	No	Python	?	?	?	
Ocrad	0.22	2013	GPL	Sí	Sí	Sí	Sí	Sí	C++	Sí	Alfabeto latino	?	

Nicomsoft OCR SDK	5.5	2015	Propietario	No	Sí	No	Sí	No	C#, VB.NET, C++, Delphi, Java	Sí	25+	?	Searchable PDF, Text, RTF
Microsoft Office OneNote 2007	?	2007	Propietario	No	Sí	No	No	No	?	?	?	?	
Microsoft Office Document Imaging	Office 2007	2007	Propietario	No	Sí	No	No	No	?	?	?	?	
MeOCR	1.0.0	2012	Free	No	Sí	No	No	No	C/C++/C#	Sí	28	Cualquier fuente impresa	HTML, hOCR, nativo, RTF, TeX, TXT
MathOCR	0.0.3	2015	GPL	No	Sí	Sí	Sí	Sí	Java	?	?	?	HTML, LaTeX
LEADTOOLS	18.0	2013	Propietario	Sí	Sí	Sí	Sí	No	C/C++, .NET, Objective-C, Java, JavaScript	Sí	56	Cualquier fuente impresa	PDF, PDF/A, DOC, DOCX, XLS, XPS, RTF, HTML, ANSI Text, Unicode Text, CSV
GOCR	0.50	2013	GPL	Sí	Sí	Sí	Sí	Sí	C	?	?	?	
FreeOCR	4.2	Agosto 2012	Propietario	No	Sí	No	No	No	?	?	?	?	
ExperVision TypeReader & RTK	7.1.170. 1125	2010	Propietario	Sí	Sí	Sí	Sí	Sí	C/C++	Sí	21	2618	
Dynamsoft OCR SDK	8.2	2012	Propietario	Sí	Sí	No	No	No	C/C++	Sí	40+	?	Texto plano
CuneiForm	12	2007	BSD variant	No	Sí	Sí	Sí	Sí	C/C++	Sí	28	Cualquier fuente impresa	HTML, hOCR, nativo, RTF, TeX, TXT

Asprise OCR SDK	5	2014	Propietario	Sí	Sí	Sí	Sí	Sí	Java, C#,VB.NET, C/C++/Delphi	Sí	20+	?	Texto plano, searchable PDF, XML
AnyDoc Software	?	?	Propietario	No	Sí	No	No	No	VBScript	?	?	?	
AliusDoc AD-SCI	2.1	2015	Propietario	No	Sí	No	No	No	VB.Net	Para extensiones	All ASCII-lenguages compatibles	?	XML,Texto plano, cualquier otra de las extensiones SDK
ABBYY FineReader	12	2014	Propietario	Sí	Sí	Sí	Sí	Sí	C/C++	Sí	198	?	DOC, DOCX, XLS, XLSX, PPTX, RTF, PDF, HTML, CSV, TXT, ODT, DjVu, EPUB, FB2
(a9t9)FreeOCR	1.022	2015	GPL	Sí	Sí	No	No	No	C#	Sí	23	Cualquier fuente impresa	TXT

Tabla II-1 Comparativa librerías OCR

2.4 Autómatas programados

Un autómata programado [Flasiński, 2002], [Flasiński, 1995] se puede definir como una tupla $Ap= (Q, I,O,M, \delta, q_0)$

Siendo:

Q : conjunto finito de estados

I : conjunto finito de estructuras de entrada

O : conjunto finito de estructuras de producción

M : conjunto finito de objetos en memoria de trabajo

$\delta: Q \times I \times \Pi \rightarrow Q \times O$: función de transición

$\Pi: A_I \cup A_M \rightarrow \{TRUE, FALSE\}$: predicado transición

A_I, A_M : conjuntos de atributos de I y M (respectivamente)

$q_0 \in Q$: estado inicial.

Estos autómatas admiten modificaciones en su definición y funcionamiento dependiendo de su cometido como reconocedores. Se pueden ver como una generalización más flexible de los autómatas convencionales con la finalidad de adaptarlos al reconocimiento en aplicaciones informáticas. Como puede apreciarse en la definición, reciben estructuras y generan otras conforme transitan por los estados mediante validaciones realizadas por consultas a los datos de la memoria de trabajo y a los de las estructuras de entrada.

2.5 Técnicas de aprendizaje automático

2.5.1 Introducción

Las técnicas de aprendizaje automático son estrategias para la obtención de patrones a partir de recopilaciones de datos. Estos datos se suelen presentar a los algoritmos como ejemplos de aprendizaje.

Dependiendo de que estos ejemplos estén ya clasificados a priori o no, se pueden dividir estas técnicas en dos categorías: técnicas supervisadas y técnicas no supervisadas [Weiss y Indurkhya, 1998]. En las técnicas supervisadas se conoce la clase a la que pertenece cada ejemplo de aprendizaje.

2.5.2 **Técnicas de inducción reglas**

La inducción de reglas es una técnica que recibe la información como un conjunto de casos (como ejemplos de aprendizaje). Estos ejemplos se representan por un conjunto de atributos común que incluye el atributo de clase. Los valores de estos atributos distinguen unos casos de otros.

Las técnicas de inducción de reglas a partir de los datos de entrada generan un árbol de decisión o un conjunto de reglas que proporcionará la clasificación de los nuevos ejemplos [Hong et al., 1986; Clark y Niblett, 1989].

Existen dos estrategias principales para conseguir la inducción de reglas:

1. Generación de un árbol de decisión y posterior extracción de sus reglas [Quinlan, 1993].
2. Aplicación de una estrategia de *covering* que genere reglas que cubran todos los ejemplos de una única clase y tras eliminar los ejemplos cubiertos proseguir con otra clase.

Un ejemplo de aplicación de la primera estrategia es el sistema C 4.5 [Quinlan, 1993], una extensión del ID3 [Quinlan, 1986], que puede generar reglas previa generación de un árbol de decisión.

Otros algoritmos como el PRISM [Cendrowska, 1987] se basan únicamente en una estrategia de *covering*, mientras que algoritmos como el PART [Frank y Witten, 1998] combinan ambas estrategias.

- Entre las ventajas que presentan estas técnicas destacan:
- La robustez frente al ruido (debidos a errores, omisiones o insuficiencia de datos).
- Identificación de atributos irrelevantes.
- Detección de la ausencia de atributos discriminantes y de vacíos de conocimiento.
- Extracción de reglas fáciles de entender y de gran expresividad.
- Posibilidad de reprocesar las reglas mediante el conocimiento de expertos, interpretando, modificando o aceptando reglas [Major y Mangano, 1995].

2.5.3 **Conjuntos de Clasificadores**

Los clasificadores son modelos que a partir de los valores de las características que representan a un elemento proporcionan la categoría a la que pertenece.

El objetivo de los conjuntos de clasificadores es mejorar la precisión que obtendrían de forma individual cada uno de los clasificadores pertenecientes al conjunto. Las decisiones que toma cada clasificador frente a un nuevo ejemplo son combinadas obteniéndose una única decisión final [Dietterich, 1997].

Entre las técnicas de construcción de conjuntos de clasificadores más comunes destacan: los clasificadores homogéneos y los clasificadores heterogéneos.

2.5.3.1 Clasificadores homogéneos

Los clasificadores homogéneos son generados a partir del mismo algoritmo de aprendizaje [Dietterich, 2000]. Los principales métodos de construcción de clasificadores homogéneos son *Bagging* [Breiman, 1996] y *Boosting* [Schapire, 1990].

Ambos métodos generan diferentes hipótesis mediante manipulación de los ejemplos de entrenamiento. El algoritmo base es entrenado con diferentes conjuntos de instancias construyéndose de esta forma cada uno de los clasificadores que forman parte del conjunto. La decisión sobre qué clasificadores proporcionan la predicción más oportuna, dependiendo de la instancia a clasificar, se realiza mediante un sistema de votos.

El éxito de estos métodos depende de manera directa de la diversidad de hipótesis generadas. Por tanto, funcionan mejor cuando el algoritmo base es un algoritmo de aprendizaje inestable, es decir, el algoritmo base construye modelos muy diferentes al presentarle distintos conjuntos de entrenamiento. Por ejemplo, los algoritmos de inducción de reglas son algoritmos de aprendizaje inestables y por consiguiente son adecuados como algoritmo base.

En concreto, *Bagging* (*Bootstrap Aggregation*) [Breiman, 1996] obtiene diferentes muestras a partir del conjunto de ejemplos de entrenamiento. Cada una de estas muestras contiene el 63,2% de instancias del conjunto original, en promedio, alcanzando el número de ejemplos total mediante repeticiones de instancias. El sistema de votos que unifica las predicciones de los clasificadores del conjunto está basado en la elección de la clase más votada por los clasificadores del conjunto.

En cambio *Boosting* [Schapire, 1990] en su versión más representativa (*AdaBoost*) [Freund y Schapire, 1995, 1996] construye los clasificadores de forma secuencial, centrándose especialmente en los ejemplos que han sido clasificados de forma errónea por el último clasificador generado. Cada ejemplo de aprendizaje recibe la asignación de un peso en función de la dificultad que presenta al intentar clasificarlo acertadamente. Estas ponderaciones son actualizadas conforme evoluciona el algoritmo. Se utiliza en este algoritmo una estrategia de voto ponderado con el fin de discernir la decisión final. El peso de cada clasificador en la votación está en concordancia con la precisión que obtiene sobre el conjunto de entrenamiento utilizado en su generación. Se tiene en cuenta en este proceso los pesos de las instancias.

2.5.3.2 Clasificadores heterogéneos

Los clasificadores heterogéneos se generan a partir de distintos algoritmos de aprendizaje. El método más utilizado en la construcción de este tipo de clasificadores es *Stacking* (*Stacked Generalization*) [Wolpert, 1992]. Este método combina las predicciones de un conjunto de clasificadores de diferente tipología mediante otro algoritmo de aprendizaje. Es decir, se implanta un clasificador en un nivel superior cuyo cometido es aprender a combinar los resultados del resto de los clasificadores.

El problema más común en la aplicación de *Stacking* es la elección de los algoritmos de aprendizaje para obtener la clasificadora base y el meta-clasificador.

2.6 Trabajos previos asociados

En el pasado se han encontrado varias aproximaciones relevantes al problema descrito, pero con objetivos o campos totalmente diferentes. Estos trabajos suelen valerse de técnicas de sketching (la traducción al castellano podría ser croquis o boceto), o lo que es lo mismo, el reconocimiento de formas a partir de entradas proporcionadas por el usuario, de manera que el computador puede asemejar lo que ha introducido el ingeniero a formas ya predefinidas.

Uno de los primeros experimentos fue el llamado Wizard of Oz [Hse et al., 1999], en el que quedó patente la preferencia de los usuarios por una interfaz basada en realizar representaciones de elementos gráficos, mediante dibujos de un sólo trazo, sobre una whiteboard, frente a las herramientas para el diseño de diagramas UML en las que se utiliza el ratón para seleccionar las diferentes estructuras. Después de mostrar su

aprobación sobre la versión de un único trazo, los usuarios expresaron su deseo de obtener algo más de flexibilidad en el uso, como por ejemplo la posibilidad de dibujar las figuras a partir de varios trazos. Carece de información concluyente acerca de dichas pruebas, por lo que no se puede hablar de éxito real en cuanto a porcentajes.

Con una finalidad semejante en el proyecto Ideogramic UML [Damm et al., 2000a] se creó una herramienta para el diseño de diagramas UML mediante la introducción de grafías, requería igualmente que el usuario introdujera en un solo movimiento las formas descritas en su propio manual de uso. Quizá el mayor inconveniente de esta característica radica en que las formas dibujadas aceptadas por el sistema podían no asemejarse mucho al resultado de salida, ya que por ejemplo, el símbolo φ era el utilizado para introducir un actor, cuya representación en pantalla es un muñeco de palo. Su última versión es la 2.3.3, tanto de manera comercial como libre, data del 16 de septiembre de 2002, y puede ejecutarse como aplicación de escritorio en pizarras electrónicas o en Tablet PCs.

Otra herramienta de reconocimiento es la surgida del proyecto Knight [Damm et al., 2000b], que utiliza como entrada y salida una gran whiteboard de alta sensibilidad para facilitar la interactividad entre usuario y sistema, consiguiendo una mezcla entre elementos formales (los que siguen el estándar) y no formales (los dibujados por el usuario). Esta aplicación tenía como propósito añadir flexibilidad y restar complejidad a la interfaz de usuario a las herramientas CASE, además de dotar de intuición al sistema para hacer más directa la creación y edición de diagramas. El diseño inicial ha sido evaluado de manera satisfactoria en experimentos y pruebas de calidad en colaboración con otras herramientas CASE, aunque serían necesarias ciertas mejoras para corregir problemas menores, como por ejemplo la asociación y aprendizaje de los gestos correspondientes a crear algunas estructuras, o como asociar elementos formales e informales entre sí. Aun así, no se han hallado los resultados reales que midan el éxito de dichas pruebas.

En ocasión del certamen CASCON 2000, el equipo formado por Edward Lank, Jeb Thorley y Sean Chean diseñó un prototipo de sistema de reconocimiento de diagramas UML introducidos a mano mediante segmentación y reconocimiento de glifos. El lenguaje en el que se implementó era Java y podía aceptar entradas desde tablets, whiteboads y ratón. Era capaz de reconocer tres tipos de diagramas: diagramas de casos de uso, diagramas de clases y diagramas de secuencia. El prototipo consistía en un kernel o núcleo independiente de reconocimiento basado en glifos [Lank et al., 2000], y una de

las metas principales que se perseguía era producir una descripción en XML de los diagramas UML. Posteriormente, se diseñaron componentes específicos [Lank et al., 2001] con los algoritmos y notaciones que refinaban los resultados del proceso llevado a cabo en el kernel, lo que le hacía capaz de procesar otros tipos de diagramas que no fueran UML, como los químicos o los matemáticos [Lank, 2001]. No se han hallado evaluaciones de este sistema, por lo que se carece de datos porcentuales que describan su fiabilidad.

Tahuti [Hammond y Davis, 2006] fue creado como basado en técnicas de sketching multitrazo, tomando como entrada tablets o whiteboards. Este sistema es capaz de reconocer los objetos tal y como los dibujaría un usuario con lápiz y papel, no es necesario que se adecúe a una grafía preestablecida por la aplicación, ya que se guía por las propiedades geométricas del dibujo, siguiendo sus líneas, ángulos y pendientes. Se efectuaron experimentos en los que se comparaba la creación y edición de diagramas de clases utilizando Tahuti, un programa de dibujo y Rational Rose, de manera que podría obtenerse el grado de usabilidad de la aplicación. Los resultados de los mismos muestran que los usuarios prefieren tanto dibujar como editar los diagramas UML con Tahuti, ya que aporta la potencia de diseño de un programa de dibujo y la versatilidad de modificación de un editor UML. La escala de los gráficos va desde 0 (mayor complicación y menor amigabilidad) hasta 5 (más sencillez y amigabilidad)

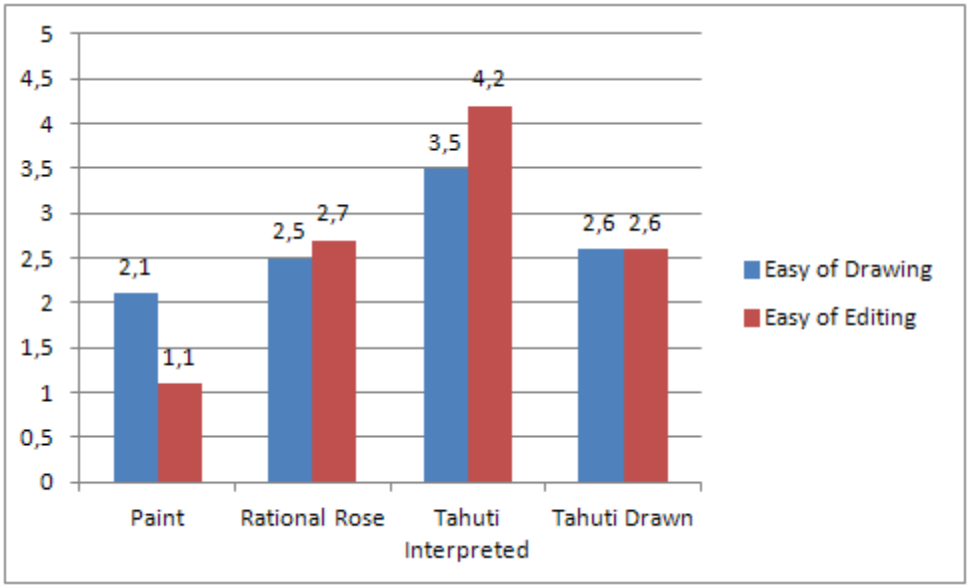


Ilustración II-28: Preferencias de dibujo y de edición

InkKit [Plimmer y Freeman, 2007] es un conjunto de herramientas de dibujo diseñado para apoyar la creación de diagramas a través de una amplia gama de dominios. Se ejecuta en Tablets PC. Se compone de una interfaz de usuario y un motor de reconocimiento adaptable. Las técnicas avanzadas de reconocimiento que se utilizan en InkKit permiten que los usuarios puedan dibujar y escribir en una página sin tener que cambiar los modos (la zona de trabajo donde se dibuja y la cartera donde se guardan los bocetos y las relaciones entre ellos). El reconocimiento de un determinado tipo de diagrama se logra mediante la creación de un dominio de diagrama y la provisión de algunos ejemplos dibujados a mano de los diferentes tipos de componentes. La salida de la aplicación puede presentarse en HTML, Java, Powerpoint o Lilypond. A continuación se muestra en la siguiente tabla los tipos de diagrama que es capaz de reconocer esta aplicación.

Domain	Interpreter			To	Output		
	Code Lines	Number of Procedures	Maximum Complexity		Code Lines	Number of Procedures	Maximum Complexity
Venn Diagram	102	2	2	Powerpoint	293	3	5
UML Class Diagram	264	4	12	Java	830	5	16
UML Activity Diagram	281	3	11	Visio			
Hierarchical Visual Model	275	2	5	HTML	255	3	6
Music	459	4	23	Lilypond	510	4	21

Tabla II-2: Tabla de formatos de entradas y salidas de InkKit

Una de las herramientas que cabe destacar de InkKit es Ink Divider [Blagojevic et al., 2010], que permite separar figuras y texto de dibujos realizados a mano a partir de reconocimiento basado en heurística. Para su desarrollo utilizaron técnicas de minería de datos para la creación de los separadores y lo entrenaron con tres tipos de diagrama: grafo dirigido, organigrama e interfaz de usuario.

Ilustración II-29: Ejemplos de diagramas de entrenamiento para Ink Divider

La aplicación de minería de datos elegida para la evaluación es Weka 3.7, que contiene un gran número de algoritmos de aprendizaje, de los que se han seleccionado Bagging, LADTree, LMT, LogicBoost, MultilayerPerceptron, RandomForest y SMO. El dataset para el entrenamiento está compuesto por 60 diagramas, de los cuales se obtienen un total de 7.248 elementos (5.616 de ellos son texto, 1.632 son figuras). Tras realizar experimentos con validación cruzada con 10 divisiones estratificadas (salvo en LADTree, por restricciones computacionales y temporales), los que mejor porcentaje de éxito arrojan son el propio LADTree (97,49%) y LogicBoost (96,70%). Para el proceso de evaluación de los algoritmos, se confrontaron los dos mejores del entrenamiento contra el antiguo clasificador [Patel et al., 2007] utilizado en InkKit y uno implementado llamado Entropy [Bhat y Hammond, 2009], utilizado como dataset 33 diagramas ER y otros tantos diagramas de proceso, lo que comportaba un total de 7.062 elementos, de los cuales 4.817 eran texto y 2.245 figuras. Los resultados vuelven a dar como vencedores a los algoritmos LADTree (95,2%) y LogicBoost (95,0%), lo que demuestra el éxito de Ink Divider para poder separar el texto de las figuras dibujadas que haya en una imagen realizada a través de un Tablet Pc.

Classifier	% Correctly classified (10-fold cross validation)
LADTree	**97.49*** (5-fold)
LogitBoost	**96.70***
RandomForest	96.45
SMO	96.41
Bagging	95.67
MultilayerPerceptron	95.02
LMT	94.85

Ilustración II-30: Resultados entrenamiento Ink Divider

Divider	% Correct	% Text	% Shapes
LADTree	95.2	98.3	88.5
LogitBoost	95.0	98.1	88.4
Old Divider	86.9	93.1	73.5
Entropy	83.3	98.7	50.5

Ilustración II-31: Resultados confrontación Ink Divider

El proyecto SILK [Landay y Myers, 1995] dio como fruto una herramienta que permite a los usuarios diseñar interfaces a través de lápiz y pad electrónicos, combinando así los beneficios de la introducción a mano con la posibilidad de establecer anotaciones y realizar modificaciones mediante gestos de edición. Ha sido evaluada por estudiantes y diseñadores profesionales, con un alto grado de aceptación, destacando su eficacia tanto para diseño creativo inicial como para las modificaciones que se puedan ir produciendo con el paso del tiempo.

Una consecuencia de SILK fue la aparición de DENIM [Lin et al., 2001], que es capaz de reconocer cajas y dos tipos de conexiones durante el diseño de páginas web por parte de los usuarios, utilizando, al igual que SILK, lápiz y pad electrónicos. Los tipos de conexiones no se diferencian por sus propiedades geométricas, sino por los objetos que conectan. Además, permite realizar tareas de sketching a diferentes niveles de refinamiento o unificarlos mediante zoom. Los experimentos contaban con siete diseñadores profesionales que mostraron las ventajas de utilizar esta herramienta con el objetivo de aumentar la productividad disminuyendo el tiempo y los errores.

Se han realizado algunas aproximaciones para el reconocimiento de diagramas en otros campos, como por ejemplo en la química, ya que mucha información se da en diagramas para describir moléculas. Se realizó un prototipo con el nombre de MolRec [Sadawi et al., 2012] para el reconocimiento de estos diagramas moleculares, que utilizando técnicas de thinning, el algoritmo Douglas-Peucker y reconocimiento basado en reglas, consigue obtener una representación gráfica de la molécula. Los resultados de los primeros experimentos con dos grandes conjuntos de diagramas son prometedores, ya que alcanza unas tasas de acierto del 88,46% y 83,84%, además de conseguir un 95% y un 94,9% de éxito en el reconocimiento de dos tramos de 1.000 diagramas procedentes del Trec 2011 [Sadawi et al., 2011].

Otro de los proyectos relacionados con la química fue Kekulé [McDaniel y Balmuth, 1992], programa creado para satisfacer la necesidad de almacenar estructuras químicas en bases de datos u otros formatos computacionales que permitan la posibilidad de edición y búsqueda. La aplicación era capaz de escanear una imagen y, de forma automática, leer las etiquetas de los átomos (mediante reconocimiento óptico de caracteres) y sus interconexiones (con ayuda de rastreo de líneas y lógica basada en reglas), para crear salidas en formato ISIS, MOLfile, SMILES, ROSDAL y otros formatos internos. Además, el software cuenta con conexiones a bases de datos externas.

Un enfoque muy parecido al anterior es el que se siguió en el proyecto CLiDE [Ibison et al., 1993], a partir del cual se desarrolló su evolución CLiDE Pro [Valko y Johnson, 2009]. Ambos productos están implementados en C++ y admiten entradas a través de scanner. La idea de extraer automáticamente información química y estructural de las imágenes de moléculas disponibles se consigue en tres fases. Primero, la imagen se segmenta en regiones de texto y gráficos. A continuación, se analizan las regiones gráficas. Finalmente, se interpretan las estructuras genéricas comparando los grupos que se encuentran en la estructura de diagramas con los que se encuentran en el texto. El resultado se presenta al usuario gráficamente para su verificación y su guardado en disco.

Existen trabajos más recientes más cercanos al cometido de esta investigación.

El objetivo del trabajo [Karasneh y Chaudron, 2013a] es extraer la información inherente a la representación de diagrama UML en formato imagen. Aunque basan su utilidad final en evitar tener que volver a dibujar el diagrama con una herramienta CASE. Indican la necesidad de reconocer formas, símbolos, líneas y texto.

Alegando pruebas experimentales deciden no reconocer líneas de longitud inferior a 20 píxeles. También presenta otras limitaciones en cuanto a la variedad del tipo de formas que se pretenden identificar. Las únicas formas complejas reconocibles son el rectángulo y las relaciones de asociación, generalización, dependencia y realización, todas ellas mediante la librería de uso gratuito Aforge.NET.Framework (http://www.aforgenet.com/framework/).

En el reconocimiento de texto utilizan el OCR MODI aunque sin indicar porcentajes de éxito. Mostrando la información combinada (estructural y textual) en formato XMI.

La experimentación la realizan con un conjunto de 10 imágenes obteniendo los siguientes porcentajes de acierto: clases 100%, relaciones 97%, símbolos 85%. Aunque indican la

existencia de falsos positivos. Además, no se dan porcentajes por tipo de relación, por lo que se deduce que evalúan la existencia o no de algún tipo de relación pero sin especificar su categoría concreta.

Entre los problemas que encontraron destacan:

- Los degradados que dan lugar a falsas relaciones reflexivas.
- El reconocimiento defectuoso del texto con OCR.
- La detección de líneas oblicuas y líneas discontinuas incompleta o incorrecta.

Por último, se echa en falta evaluaciones sobre el tiempo de proceso.

Los mismos autores rehicieron el trabajo anterior [Karasneh y Chaudron, 2013b]. Como novedades centraron el objetivo en la posibilidad de crear repositorios de modelos. Además, amplían el conjunto de experimentación a 200 imágenes obteniendo los siguientes resultados en el porcentaje de elementos reconocidos: clases 95%, relaciones 80%, texto 92%. Sin embargo, persisten el resto de deficiencias (no hay evaluación por tipo de relación, limitación a la forma rectangular, no hay mediciones de tiempo de ejecuciones y se mantienen los problemas descritos anteriormente).

La clasificación de imágenes como representación o no de un diagrama de clases fue el objetivo perseguido en otro trabajo [Hjaltason y Samúelsson, 2015]. En este caso se utilizó la biblioteca libre de visión artificial OpenCV (http://opencv.org/) combinada con, la biblioteca de procesamiento de imágenes de código abierto, Magick++ (http://www.imagemagick.org/Magick++/) para transformar los formatos no soportados por OpenCV a otros compatibles.

Se practica una preselección atendiendo al color más frecuente (al menos en el 10% de la imagen) y al valor medio del histograma de color que debe estar por encima de 100. El objetivo de la preselección es eliminar imágenes fotográficas y otras que no representen diagramas. Sin embargo, no se dan datos sobre la evaluación de este filtrado previo de imágenes.

Después, se practica sobre las imágenes no eliminadas una detección de líneas y formas muy costosa, utilizando varios métodos, para asegurar detección por redundancia, además de mediciones de ángulos y uso de ecuaciones para las formas.

No se hace análisis textual por lo que puede obtener como diagramas de clases otros con idéntica estructura. Resuelven este aspecto basándose en el tipo de consultas web efectuadas, en google image indicando un dominio, con el fin de obtener imágenes positivas (representan diagramas UML).

Se obtienen modelos de aprendizaje automático a partir de 650 imágenes positivas y otras tantas negativas, obtenidas estas últimas mediante recopilación de otros tipos de diagramas, gráficos, planos y mapas.

Se utilizan 23 atributos (se descartó la selección de atributos) a partir de valores extraídos para la clasificación. Los resultados de evaluación con máquinas vectoriales son en media del 92% de acierto. Se realizaron tres pruebas tomando el 60% de los ejemplos para el entrenamiento y el 40% para el test.

La principal limitación radica en el tiempo medio de procesado por imagen que alcanza los 10 segundos. Lo que implica que no es un procedimiento adecuado para incorporarlo, con miras a una optimización temporal, como paso previo a un procesado de extracción de información sobre imágenes que pudiesen contener diagramas UML.

Se ha encontrado una tesis de licenciatura [Van den Bosch, 2014] en la que se ha creado un software que es capaz de procesar archivos binarios de imágenes que contengan diagramas de casos de uso para poder generar con su contenido un archivo con extensión XMI, que puede ser importado y editado en Visual Paradigm. Los resultados del estudio muestran buenos resultados en la conversión de los diagramas, aunque también se ven ciertas deficiencias en el software. La aplicación está desarrollada en C++, y acepta imágenes con extensión .JPG y .PNG. Se han realizado tres tipos de test, en los que, sobre todo el primero, queda patente que el alto grado de acierto, pero no en los otros dos, lo que hace ver que el software no es aún lo suficientemente robusto. Cada test contiene un conjunto de imágenes que han pasado un criterio de valoración que depende de si los elementos que contienen están representados conforme a figuras estándar o presentan formas alternativas.

Testset1: Contienen imágenes que han pasado el criterio de valoración.

Testset2: Contienen imágenes que pasan el criterio de valoración, pero también se han incluido uno o dos elementos que no lo han pasado.

Testset3: Contienen imágenes con dos o más elementos que no pasan el criterio de valoración.

	Testset 1		Testset 2		Testset 3	
	Mean (%)	Stdev (%)	Mean (%)	Stdev (%)	Mean (%)	Stdev (%)
Rec. Use cases	92	18	81	31	74	35
Rec. Actors	89	29	74	35	38	44
Rec. Relations actor-usecase	69	36	48	39	21	32
Rec. Relations usecase-usecase	85	17	47	37	57	31
Rec. System border	92	26	88	33	81	46

Tabla II-3: Resultados de test de reconocimiento de casos de uso

Como resumen final, y con el objetivo de tener una visión más general y resumida de los trabajos mencionados en este apartado, se han recopilado sus características representativas en la siguiente tabla:

	Entrada	Salida	Dispositivos usados	Reconocimiento	Observaciones
Wizard of Oz	Diagramas UML dibujados VS realizados con editor	Diagramas UML	Whiteboard, Ratón	Sketching monotrazo	Experimento para enfrentar el diseño con whiteboard frente al diseño con ratón
Ideogramic UML	Diagramas UML dibujados	Diagramas UML	Ratón, Whiteboard, Tablet PC	Sketching monotrazo	Entorno de trabajo para Java y C++. Uso de glifos propios
Proyecto Knight	Diagramas UML dibujados en whiteboard	Diagramas UML en whiteboard	Whiteboard	Sketching monotrazo	Mezcla elementos formales y no formales.
Prototipo E. Lank	Diagramas UML dibujados	Diagramas dependientes del dominio del componente	Ratón, Whiteboard, Tablet PC	Segmentación y reconocimiento de glifos	Realizado en Java. Prototipo, no hay experimentación
Tahuti	Diagramas UML dibujados	Diagramas UML	Whiteboard Tablet PC	Scketching multitrazo	Mejor valorado que Rational Rose. Para dibujo y edición

InkKit	Dibujos de los tipos de diagrama admitidos	Diagramas de Ven, Diagramas de Clases y Actividad en UML, Modelo Jerárquicos, Partituras	Tablet PC	Motor de reconocimiento basado en componentes específicos	Formato de salida en HTML, Java, Powerpoint o Lolypond
Ink Divider	Dibujo de interfaz de usuario	Texto y figuras por separado	Tablet PC	Reconocimiento basado en heurística	Éxito en torno al 95% en los experimentos realizados
Silk	Dibujo de interfaz de usuario	Interfaces de usuario operativos	Whiteboard	Scketching multitrazo	Diferencia las conexiones por sus propiedades geométricas
Denim	Dibujo de interfaz de usuario	Interfaces de usuario operativos	Whiteboard	Sketching multinivel	Diferencia las conexiones por los objetos conectados.
MolRec	Imágenes de moléculas	Imágenes editables de moléculas con información	Imágenes binarias	Técnicas de thinning, algoritmo Douglas-Peucker y reconocimiento basado en reglas	Tasas de acierto superiores a 88 y 83% en experimentos, y llegan al 95% con diagramas del Trec 2011
Kekulé	Imágenes de moléculas	Archivos ISIS, ISIS, MOLfile, SMILES, ROSDAL y Bases de datos	Scanner	Reconocimiento óptico de caracteres, seguimiento de líneas y lógica de reglas	Además de los formatos mencionados también puede guardar en formatos internos

CLiDE	Imágenes de moléculas	Gráficas y Bases de datos	Scanner	Segmentación y reconocimiento de figuras	Realizado en C++. Utiliza técnicas de inteligencia artificial
CLiDE Pro	Imágenes de moléculas	Gráficas y Bases de datos	Scanner	Segmentación y reconocimiento de figuras	Realizado en C++. Evolución de CLiDE
Karasneh y Chaudron, 2013a	Imágenes con diagramas de clases	Archivos con extensión XMI	Imágenes binarias	Reconocimiento por bibliotecas de código	Experimentación realizada con 10 imágenes. No hay evaluaciones de tiempo de proceso, ni del OCR, ni del tipo de relaciones.
Karasneh y Chaudron, 2013b	Imágenes con diagramas de clases	Archivos con extensión XMI	Imágenes binarias	Reconocimiento por bibliotecas de código	Experimentación realizada con 200 imágenes. No hay evaluaciones de tiempo de proceso ni del tipo de relaciones.
Hjaltason y Samúelsson, 2015	Imágenes con diagramas	Clasificación de imágenes	Imágenes binarias	Reconocimiento por bibliotecas de código	92% de acierto. Tiempo medio de procesado por imagen de 10 segundos.
Van den Bosch, 2014	Imágenes JPG y PNG con diagramas de casos de uso	Archivos con extensión XMI visibles desde Visual Paradigm	Imágenes binarias	Reconocimiento por bibliotecas de código	Realizado en C++. Aún poco robusto, máximo 92% de acierto en test ideales.

Tabla II-4: Tabla resumen de trabajos previos

Capítulo III: Desarrollo de la Investigación y Marco Experimental

Como se planteaba en la introducción de este trabajo, el objeto del presente estudio es establecer un método de extracción automática de la información, tanto estructural como textual, de las imágenes de diagramas UML. Incluyendo las de baja resolución por su procedencia web. Con este objetivo se han marcado tres fases de investigación, desarrollo e experimentación que se simplifican en la siguiente Ilustración III-1:

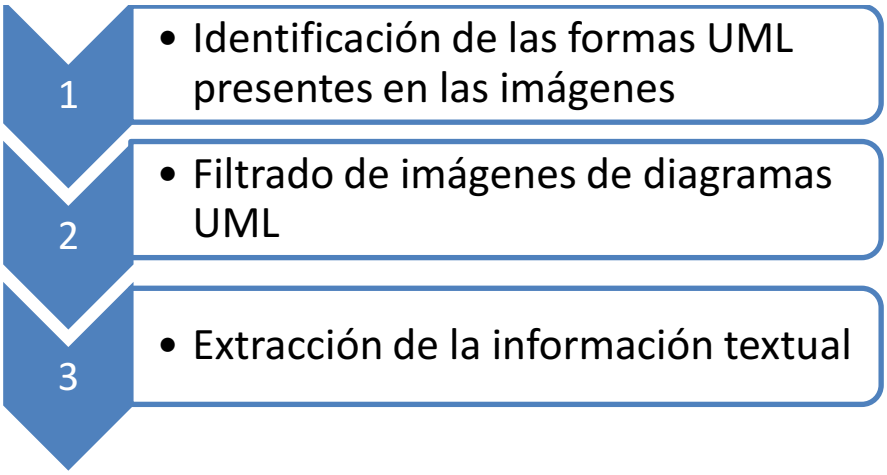


Ilustración III-1: Fases de la investigación

La fase 1 consiste en la identificación de las formas presentes en las imágenes de diagramas UML. Para ello se localizarán los píxeles más representativos y se detectaran las figuras progresivamente de las más simples a las más complejas.

En la fase 2 se investiga el filtrado de imágenes como forma de optimización de procesos masivos en los que intervienen multitud de imágenes sin garantías de que todas ellas contengan diagramas UML.

Por último, en la fase 3 se extraerá la información textual de las imágenes mediante aplicaciones OCR. Aplicando y adaptando estudios previos se determinarán algoritmos capaces de reconocer caracteres en imágenes con resoluciones inferiores a las recomendadas por las herramientas OCR.

Estas fases responden al planteamiento de definir su metodología, realizar la experimentación y evaluar los resultados para su validación.

3.1 Detección de formas UML

En este apartado el objetivo principal es la identificación o detección de formas o figuras que representan elementos UML. En el epígrafe 1.7 se explicó cómo serían las imágenes de trabajo en cuanto a su formato e imágenes contenidas. Tras las comprobaciones oportunas con el fin de determinar que la imagen a procesar pertenece a alguno de los casos de estudio se seguirán los siguientes pasos:

1) Umbralización
2) Detección de bordes
3) Detección de segmentos verticales y horizontales
4) Detección de formas delimitadas por los segmentos anteriores
5) Detección de segmentos oblicuos
6) Detección de enlaces
7) Asignación de la tipología de los enlaces por detección de sus cabezas

3.1.1 **Umbralización**

La técnica de umbralización se ha aplicado a las imágenes con el fin de distinguir los píxeles que representan objetos de aquellos que conforman el fondo. El resultado será una imagen binaria formada por píxeles negros para los objetos y blancos para el fondo.

El conjunto de características a comparar para determinar si dos píxeles son semejantes (criterios de homogeneidad) depende fundamentalmente de la presencia de sombras en la imagen y en la curvatura de la superficies [Gevers y Groen, 1991].

Las herramientas de diagramación suelen proporcionar imágenes con regiones homogéneas, con pocas tonalidades de gris que diferencian fuertemente el fondo de los píxeles con información útil. La siguiente imagen es constituye un ejemplo típico:

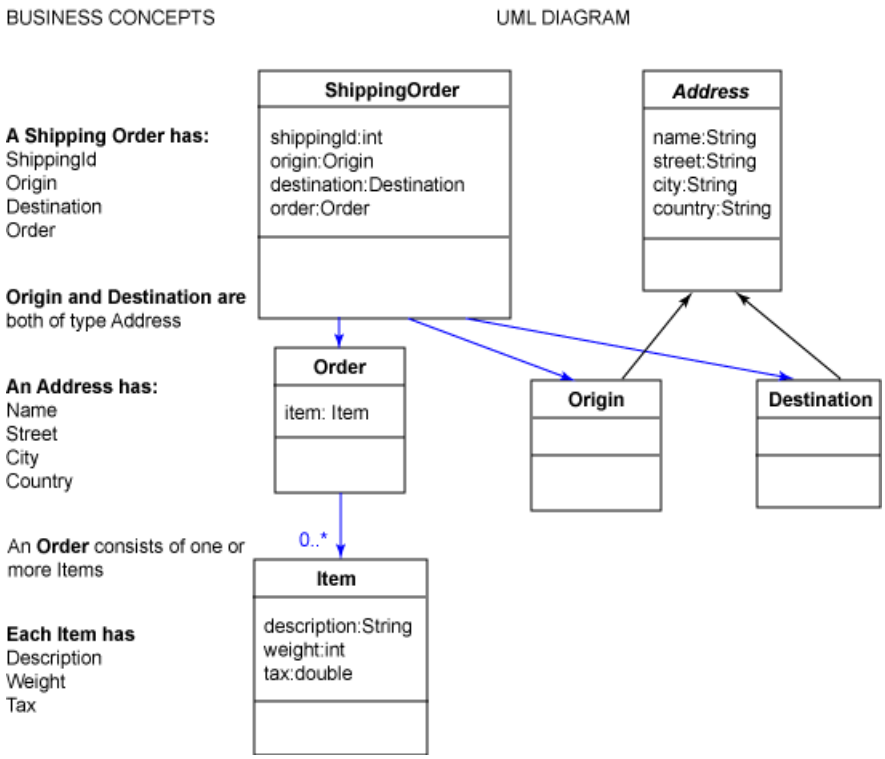


Ilustración III-2: Diagrama UML 1[6]

[6] Imagen tomada de UMLApplication/UMLApplication33.htm (Crawled on 2010-03-12)

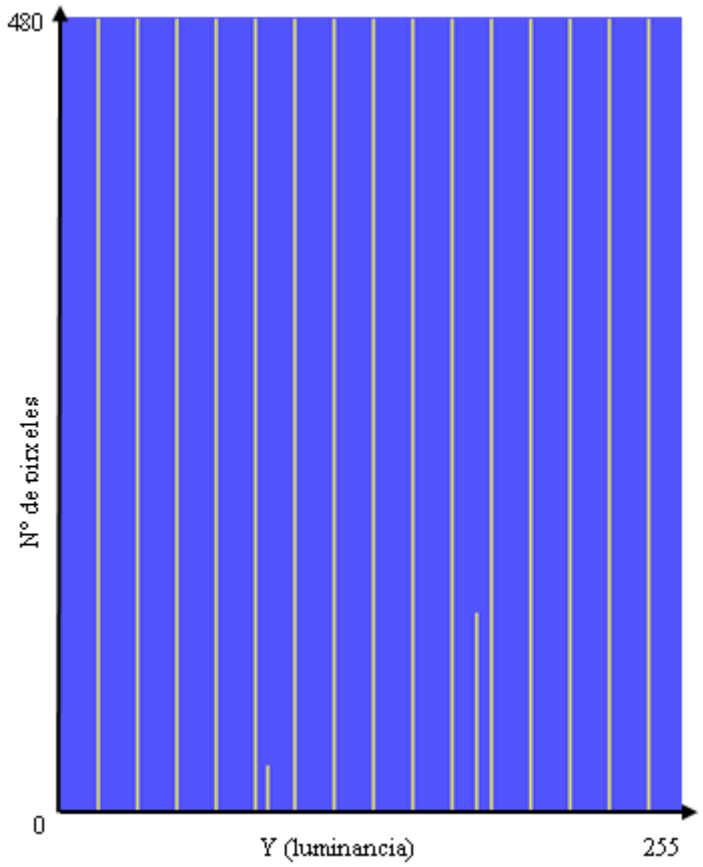


Ilustración III-3: Histograma del diagrama UML 1.

Sin embargo, es posible encontrar imágenes de diagramas que cubren todo el abanico de tonalidades de gris por incorporar sombreados u otros efectos de diseño artístico. La siguiente imagen es una muestra de ello.

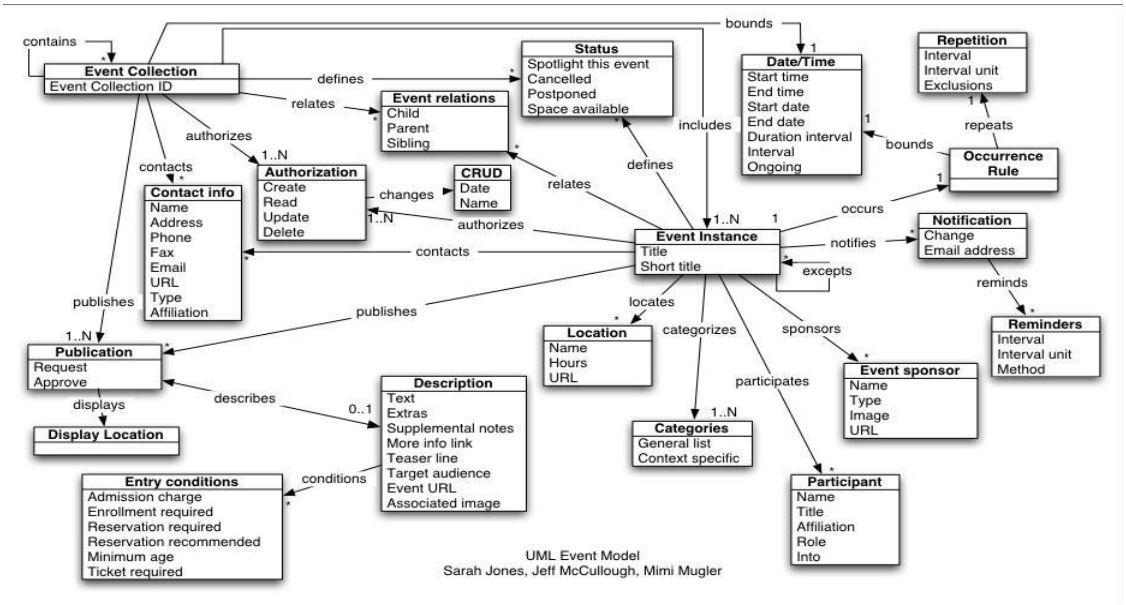


Ilustración III-4: Diagrama UML 2.[7]

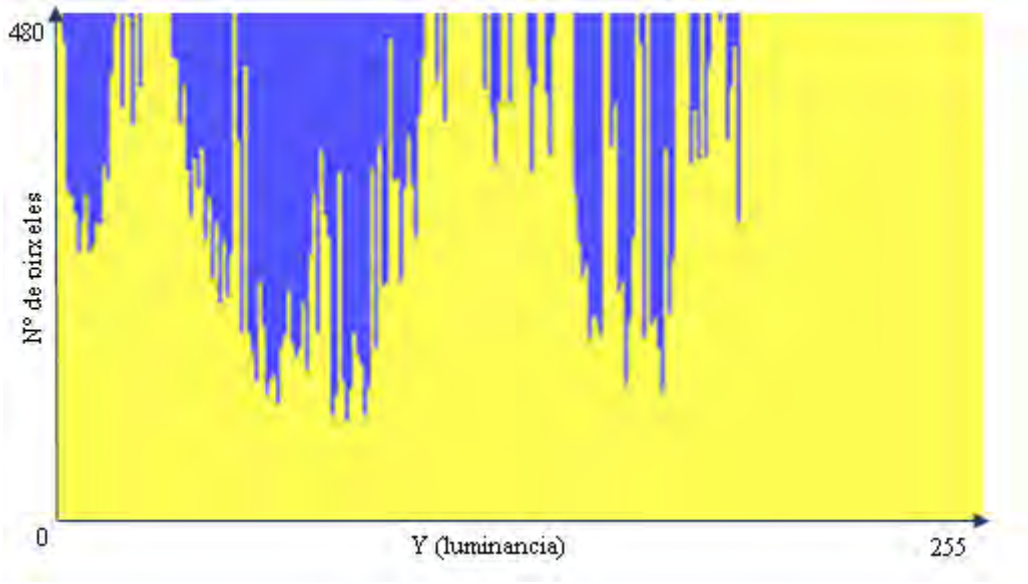


Ilustración III-5: Histograma del diagrama UML 2.

Tras analizar una colección de imágenes, con la información de los píxeles en su tonalidad de gris, se ha decidido que los píxeles adyacentes con diferencias en sus tonos de gris no superiores a 80 unidades pertenecen a una misma región.

[7] Imagen tomada de http://tweakers.net/nieuws/48673/microsoft-uml-gaat-niet-ver-genoeg.html (Crawled on 15/04/2008)

La ecuación utilizada en los cálculos de los histogramas de niveles de gris, a partir de los valores de los canales del modelo RGB, es:

[7] $$Y = 0.299R + 0.587G + 0.114B$$

Se han redondeado cada valor de *Y* calculado al entero más próximo.

3.1.2 **Detección de bordes**

Una vez aplicada la umbralización los bordes se consiguen seleccionando los píxeles negros, que tengan píxeles vecinos blancos (píxeles del fondo) en disposición vertical u horizontal. [Moreno et al., 2006]

La siguiente figura muestra dos de las 15 posibles situaciones en las que un píxel pertenece al borde. En ambas se representa un píxel negro (1) con otros adyacentes blancos (0's) en horizontal o vertical.

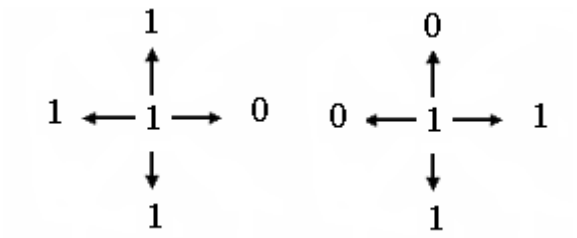


Ilustración III-6: Ejemplos de situacions en las que un píxel pertenece al borde

En la Ilustración III-6 se representan situaciones en las que un píxel no pertenece al borde. La primera recrea un píxel negro (1) que no tiene otros adyacentes blancos (0's) en horizontal o vertical. La última corresponde al análisis de un píxel blanco (0), por lo que no puede pertenecer a ningún borde.

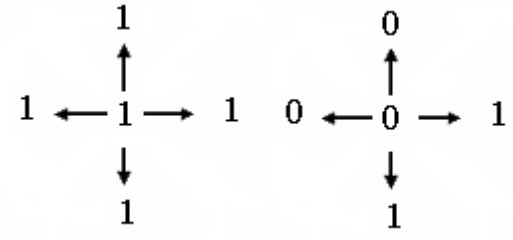


Ilustración III-7: Ejemplos de situacions en las que un píxel no pertenece al borde

La detección la umbralización y la detección de bordes se realizan en un mismo procesado de las imágenes con el fin de optimizar la eficiencia.

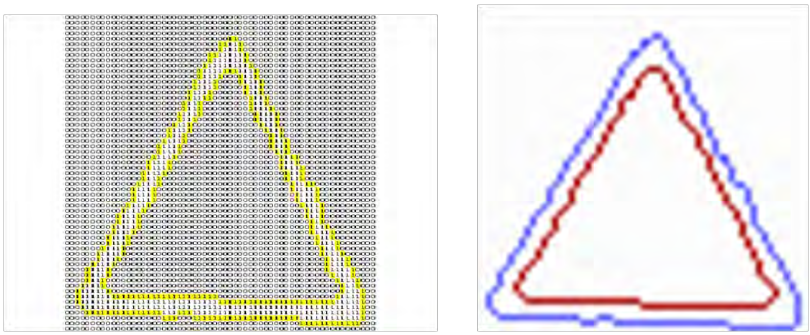


Ilustración III-8: Ejemplo de detección de bordes

3.1.3 **Detección de segmentos verticales y horizontales**

En la detección de segmentos verticales y horizontales, como candidatos a ser lados de figuras UML o relaciones entre ellos, se tendrán en cuenta solamente los puntos que han pasado el filtro de detección de bordes. El método consiste en recorrer pixel a pixel la imagen, en el caso de que se encuentre un pixel negro se comprueba primero si forma parte de una línea vertical (explorando los píxeles hacia abajo) y después si forma parte de una línea horizontal (explorando los píxeles hacia la derecha). La exploración de un segmento se detiene cuando llega a los límites de la imagen o encuentra más de 5 pixeles blancos consecutivos.

Además de ir encontrando todos los segmentos verticales y horizontales existentes en la imagen se obtienen simultáneamente las relaciones entre ellos. De esta forma, cada vez que se encuentra un pixel negro, extremo de un segmento, perteneciente a otro segmento se refleja la relación simétrica existente entre ellos. Además, para cada segmento se almacena una serie de características que serán útiles para el posterior procesado de elementos:

- **Identificador:** Número del orden en el que se ha detectado el segmento y que por tanto es único para cada segmento. La comparación de los identificadores de dos segmentos proporciona información sobre la posición relativa de los mismos en la imagen. De esta forma, por ejemplo, si un segmento *A* tiene un identificador menor que el de un segmento *B* y ambos son horizontales, implica que el

segmento *A* no puede estar en la imagen en una posición inferior a la del segmento *B*. Además, de estar a la misma altura *A* debe encontrarse a la izquierda de *B*.

En el caso de que dos segmentos compartan vértices también queda determinado el orden de los identificadores según los recorridos de exploración de píxeles.

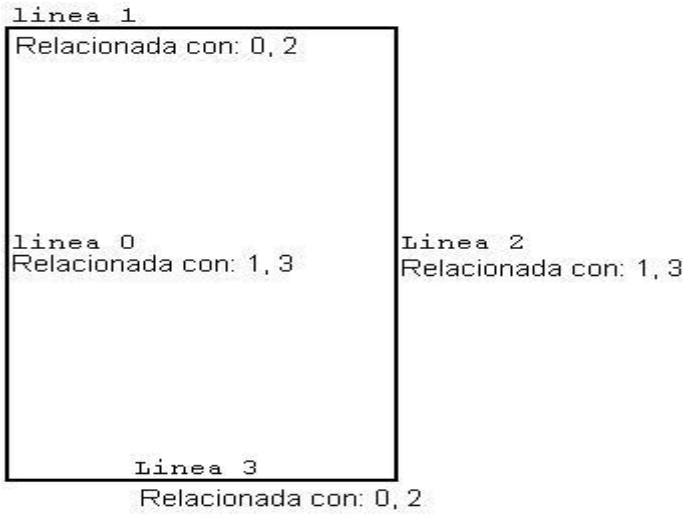


Ilustración III-9: Ejemplo de asignación, por orden de recorrido, de los identificadores

En este caso particular (Ilustración III-9) se detectan 4 segmentos. Se puede observar como el segmento que tiene asignado el identificador 0 es el que contiene un píxel en una posición no inferior al del resto de píxeles de la imagen, siendo el situado más a la izquierda de entre todos los que comparte su nivel. El segmento 1 empieza en la mismo píxel que el segmento 0 (forman un vértice), por el orden de exploración de píxeles se identifica primero el segmento que se encuentre en posición vertical. Se muestran también las conexiones entre segmentos.

- **Posición:** indica las coordenadas en las que se encuentra el segmento dentro de la imagen. De esta información se puede derivar otra como la longitud del segmento
- **Discontinuidad:** determina si un segmento es continuo o discontinuo, atendiendo a su densidad. Es decir, el cociente entre el número de píxeles negros que contiene y su longitud.
- **Relacionadas:** Indica para un segmento los segmentos conectados con él. No incluye los cruces de segmentos. Por tanto, no se establece relación entre segmentos si el píxel que comparten no es un extremo de alguno de ellos.

3.1.4 **Detección de formas principales**

Se denomina, en este trabajo, como formas principales a aquella que podrían presentar relaciones entre ellas en diagramas UML. Es decir, clases, paquetes, notas, elementos parametrizados y componentes. La estrategia a seguir será reconocerlas partir de los segmentos horizontales y verticales que se hayan detectado. Para ello se emplearan autómatas programados [Flasiński, 2002], [Flasiński, 1995] (ver epígrafe 2.4). En algunos casos los autómatas reconocerán las formas directamente y en otros, como en los paquetes, detectara formas más simples que serán combinadas para completar el reconocimiento.

Se definirá un autómata programado $Ap= (Q, I, O, M, \delta, q_0)$ por cada forma a reconocer.

Siendo:

Q: conjunto de estados

I: conjunto de segmentos verticales y horizontales detectados

O: secuencia de segmentos reconocida

M: listas de adyacencia (información de las conexiones entre segmentos) y demás datos

$\delta: Q \times I \times \Pi \rightarrow Q \times O$: función de transición

$\Pi: A_I \cup A_M \rightarrow \{TRUE, FALSE\}$: predicado transición

A_I, A_M: conjuntos de atributos de ***I*** (identificador, posición, discontinuidad) y ***M***

$q_0 \in Q$: estado inicial.

Las posibles secuencias de segmentos a reconocer se obtendrán recorriendo en profundidad el grafo implícito ***G(V,E)*** donde el conjunto de nodos ***V*** es el conjunto de segmentos (***V=I***) y el conjunto de aristas representa las conexiones entre segmentos (información contenida en ***M***).

En la Ilustración III-10 se aprecia un ejemplo del grafo implícito de conexiones de segmentos. Este grafo se recorre en profundidad de forma que las cadenas de segmentos obtenidas en la exploración ya cumplen las restricciones de conectividad requeridas por la forma a reconocer (en este caso un componente).

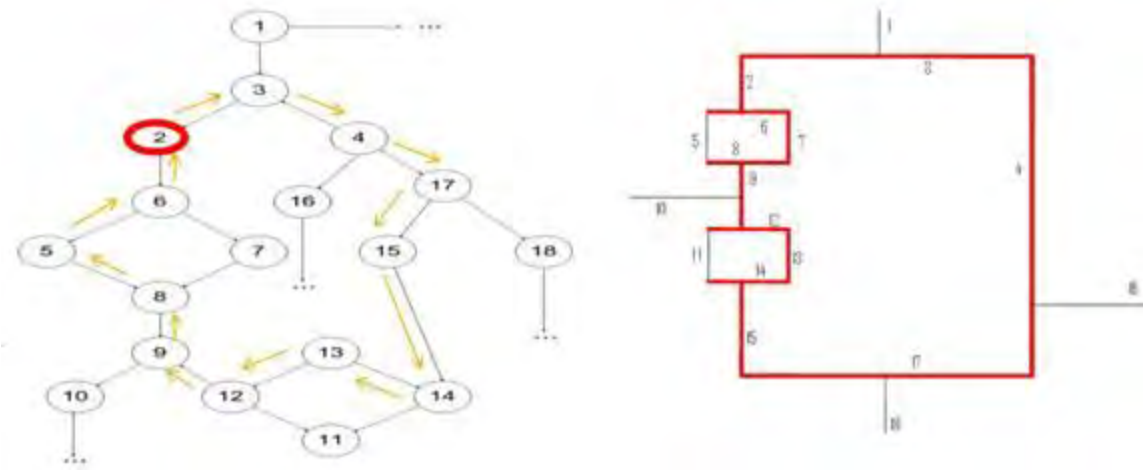


Ilustración III-10: Ejemplo del grafo implícito formado por los segmentos y sus conexiones.

Destacar que el reconocimiento de todas las formas reconocidas por autómatas programados se realiza de forma simultánea en un único recorrido del grafo implícito ***G(I,E)***. Para ello sólo se aplicará vuelta atrás en el recorrido en profundidad de una rama del grafo cuando en todos los autómatas se haya alcanzado el estado basura o sumidero. Además, en cada situación de vuelta atrás se recupera la situación en que se encontraba cada uno de los autómatas evitando así comenzar desde el principio los reconocimientos.

De esta forma los autómatas, en una sola exploración del grafo, procesan todas las cadenas de segmentos candidatas a ser reconocidas consiguiéndose una clara ventaja, en eficiencia, con respecto al uso de librerías que permiten funcionalidades de reconocimiento de aplicación individualizada y por tanto no simultánea.

3.1.4.1 Detección de rectángulos

El rectángulo en los diagramas UML es la forma utilizada para representar clases, objetos e ítems además de ser parte de otros elementos como los paquetes. La detección de rectángulos se practica a partir de los segmentos verticales y horizontales identificados según el apartado anterior. El objetivo es encontrar cuatro segmentos que se encuentren conectados secuencialmente y cuyos identificadores vayan en orden ascendente. Esta última restricción va dirigida a la representación única de un rectángulo a partir de una secuencia de sus lados, comenzando por uno vertical. Además, se exige que los segmentos tengan una longitud mínima y no sean discontinuos salvo en los casos de parametrizaciones UML.

Para tal fin se hace uso de un autómata programado ***Ap= (Q, I ,O, M, δ, q_0)*** [Flasiński, 2002], [Flasiński, 1995] que reconocen las secuencias de segmentos que cumplen las restricciones mencionadas.

En la Ilustración III-11 se representa el autómata programado que reconoce rectángulos.

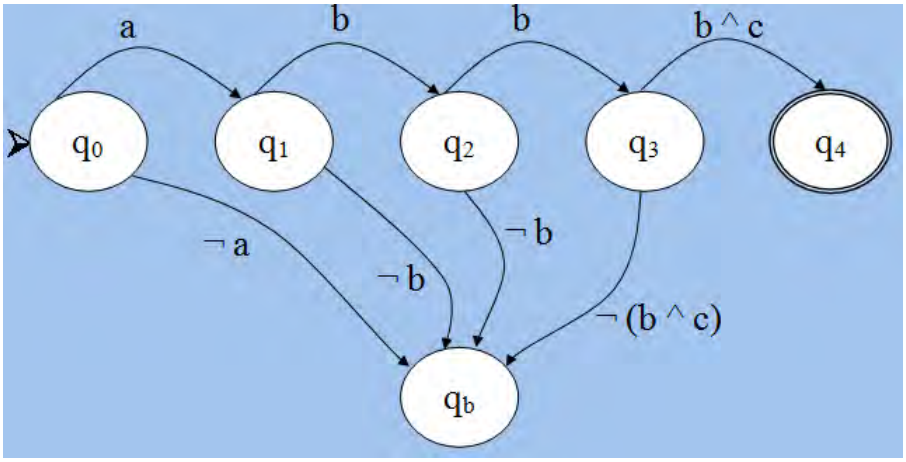


Ilustración III-11: Autómata Programado para la detección de rectángulos.

Siendo:

$a(q_i , q_j , I_n, A_I, A_M) \equiv lineaActual.esContinua \wedge lineaActual.longitud \geq 10$

$b\ (q_i , q_j , I_n, A_I, A_M) \equiv a(q_i , q_j , I_n, A_I, A_M) \wedge linea\ Anterior.id < lineaActual.id \wedge$

$\wedge\ lineaActual.concectada(lineaAnterior)$

$c(q_i , q_j , I_n, A_I, A_M) \equiv lineaActual.concectada(lineaPrimera)$

La longitud se mide en píxeles. La linea actual se corresponde con el segmento (símbolo de entrada) objeto de transición. Es decir, *lineaActual* = I_n . La linea anterior es el último segmento incorporado a la estrucutra de salida *O* al transitar al estado q_i. Es decir, *lineaAnterior* = *Último*($O(q_i)$). La linea primera es el primer segmento incorporado a la estrucutra de salida *O* acumulada al transitar al estado q_i. Es decir, *lineaPrimera* = *Primero*($O(q_i)$). Se han omitido los argumentos de *a, b* y *c* en la representación gráfica del autómata para facilitar su comprensión. Las definiciones y consideraciones expuestas se aplican también al resto de autómatas que reconocen formas principales.

En la siguiente ilustración se muestra el proceso de reconocimiento seguido por el autómata.

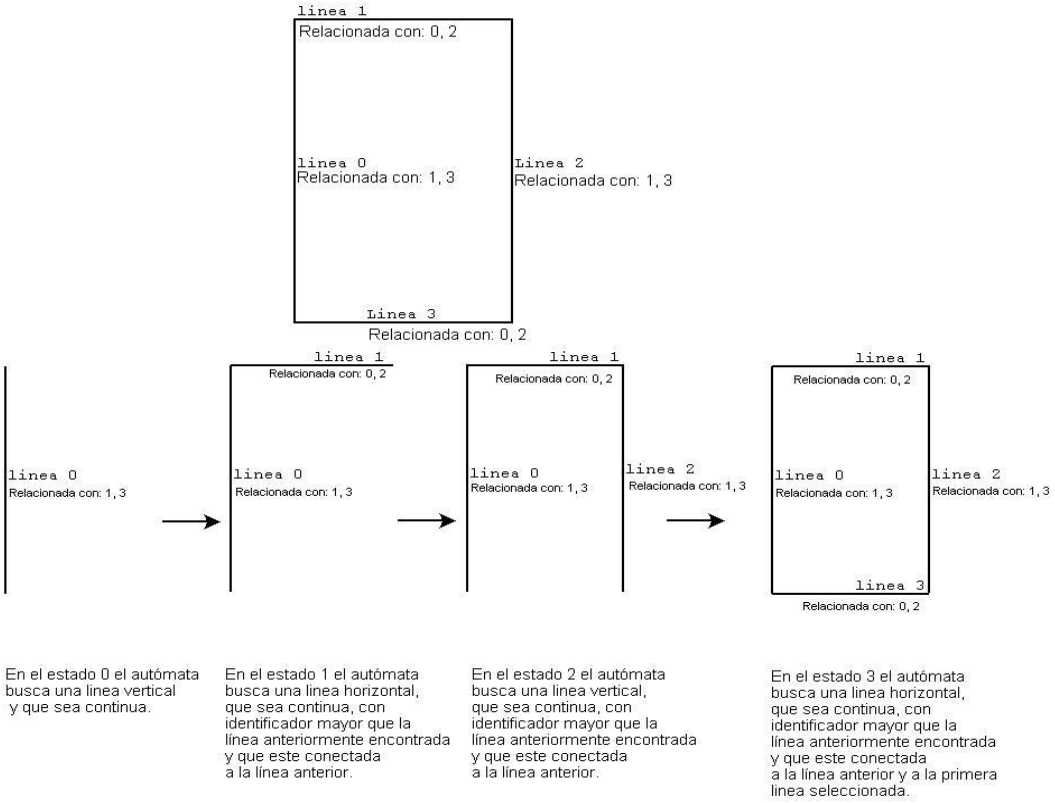
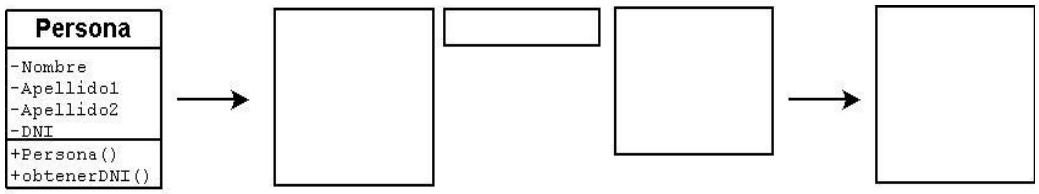


Ilustración III-12: Ejemplo de reconocimiento de un rectángulo por el autómata.

Una vez se han reconocido todos los rectángulos presentes en el diagrama, se procede a distinguir los rectángulos principales (aquellos que no están contenidos en otros). En caso de que los vértices de un rectángulo estén dentro del área de otro rectángulo se registra esa inclusión. Finalmente, los rectángulos no marcados como contenidos serán los principales.

Este proceso se hace más eficiente debido a la restricción en el orden de los identificadores de los segmentos que forman un rectángulo. De esta forma, como se puede apreciar en la Ilustración III-13, un rectángulo que contiene a otros tres independientes (una clase UML) proporciona sólo tres identificaciones de rectángulos, cuando cabe esperar seis (uno principal, dos conteniendo a dos individuales y los tres individuales).



Se observa como de una clase se pueden encontrar hasta 3 cajas distintas. El objetivo de la busqueda de cajas principales consigue que de estas 3 cajas sólo quede 1.

Ilustración III-13: Ejemplo de detección de rectángulos principales y contenidos.

Según el autómata descrito en la Ilustración III-13, los rectángulos distinguidos en rojo en la Ilustración III-14 no se pueden reconocer, ya que su tercer lado pertenece a un segmento cuyo identificador (2) es menor que el del segmento precedente (4 o5 según el caso).

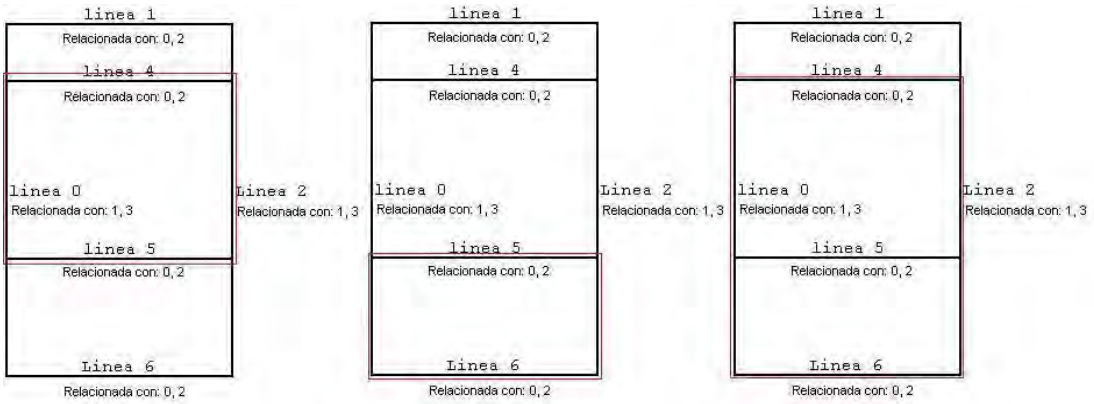


Ilustración III-14: Ejemplo de rectángulos que no se detectan con el autómata.

Esta simplificación de la casuística en las inclusiones de rectángulos aumenta la eficiencia en la identificación de rectángulos principales, aunque hay que hacer adaptaciones para distinguir si el rectángulo principal corresponde a una clase (con 1,2 o 3 divisiones), a un objeto o a un ítem. La discriminación se expresa en la siguiente tabla:

Elemento	Nº de rectángulos reconocidos	Nº de rectángulos principales	Nº de rectángulos contenidos
Clase (3 divisiones)	3	1	2
Objeto o clase (2 divisiones)	2	1	1
Ítem o clase (1 división)	1	1	0

Tabla III-1: Distinción de formas atendiendo al número de rectángulos detectados

3.1.4.2 Detección de paquetes

La detección de paquetes se consigue mediante combinación de elementos gráficos. Un paquete queda identificado si dos rectángulos tienen sus lados izquierdos en el mismo segmento y el lado inferior de uno de ellos es parte del lado superior del otro.

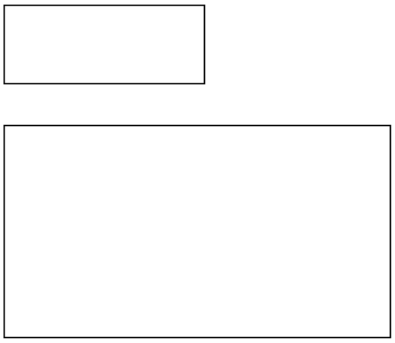


Ilustración III-15: Ejemplo de combinación de elementos gráficos para formación de paquetes.

3.1.4.3 Detección de notas

En la detección de formas que representan notas se emplea un autómata programado [Flasiński, 2002], [Flasiński, 1995] construido de manera análoga al usado en el reconocimiento de rectángulos. En este caso se busca seis segmentos que se encuentren conectados secuencialmente y cuyos identificadores vayan en orden ascendente. Esta secuencia de segmentos alterna de manera evidente verticalidad y horizontalidad. Los segmentos deben tener una longitud mínima y no ser discontinuos.

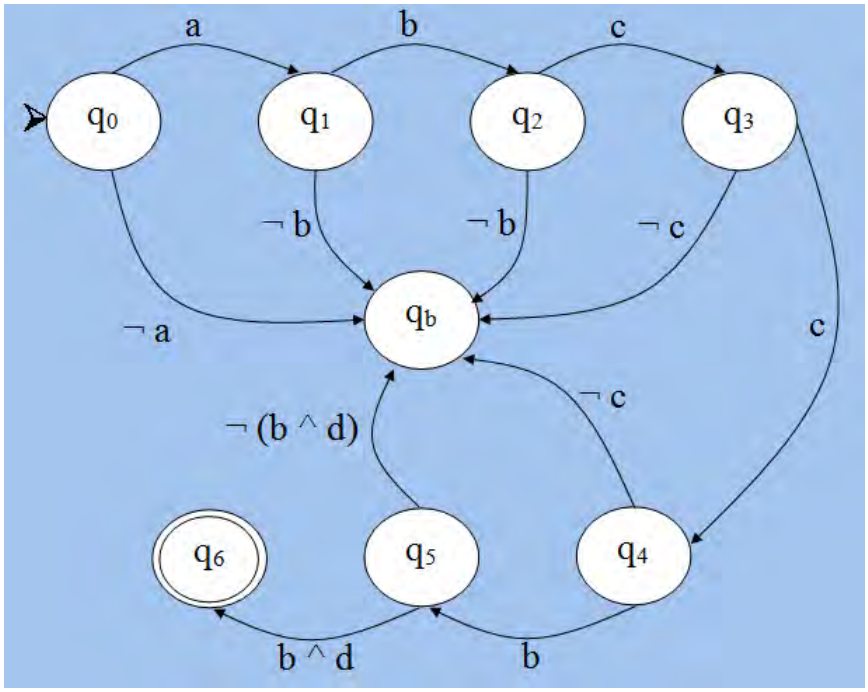


Ilustración III-16: Autómata para la detección de notas.

Donde:

$a(q_i , q_j , I_n, A_I, A_M) \equiv lineaActual.esContinua \wedge lineaActual.longitud \geq 10$

$b (q_i , q_j , I_n, A_I, A_M) \equiv a(q_i , q_j , I_n, A_I, A_M) \wedge linea\ Anterior.id < linActual.id \wedge$

$\wedge lineaActual.concectada(lineaAnterior)$

$c (q_i , q_j , I_n, A_I, A_M) \equiv lineaActual.esContinua \wedge lineaAnterior.id > linActualId \wedge$

$\wedge lineaActual.concectada(lineaAnterior)$

$d (q_i , q_j , I_n, A_I, A_M) \equiv lineaActual.concectada(lineaPrimera)$

3.1.4.4 Detección de clases, objetos e ítems parametrizados

La detección de estas formas se consigue mediante combinación de elementos gráficos. Un rectángulo con lados discontinuos y por tanto reconocible por el autómata descrito en la sección 3.1.4.1, y otra forma similar a la reconocida por el autómata de notas (Ilustración III-16) salvo por la discontinuidad de los segmentos coincidentes con los del rectángulo. Además, el segmento vertical discontinuo tiene, en este caso, un identificador menor que el de su segmento predecesor en la secuencia.

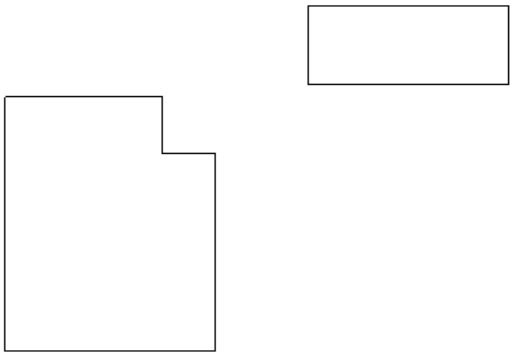


Ilustración III-17: Ejemplo de combinación de elementos gráficos para formación de clases parametrizadas.

Se muestra a continuación (Ilustración III-18) el autómata que reconoce la segunda forma de la combinación:

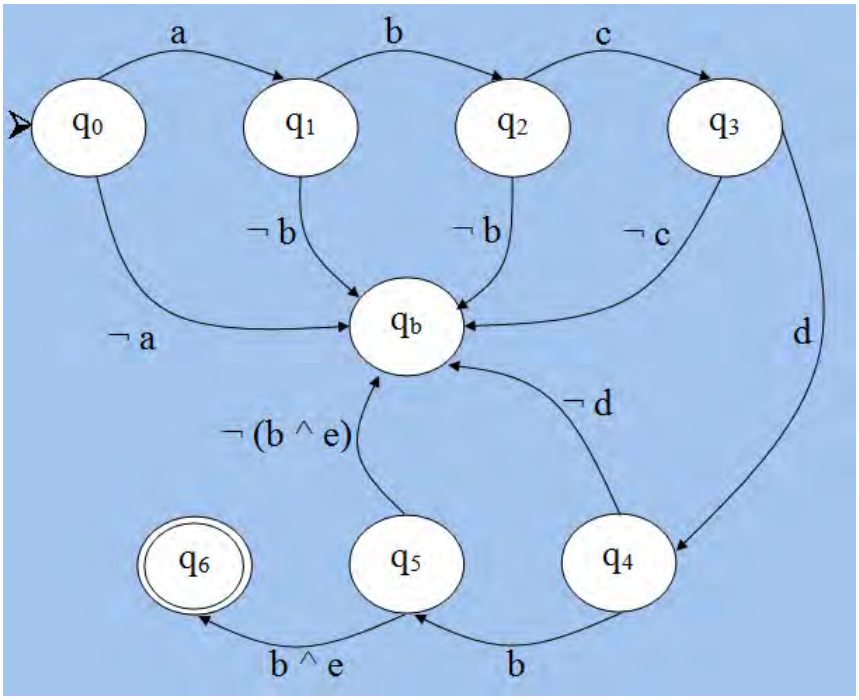


Ilustración III-18: Autómata para la detección de clases parametrizadas.

Donde a, d, c, d y e se definen a continuación:

$a\ (q_i\ ,\ q_j\ ,I_n,\ A_1,\ A_M) \equiv lineaActual.esContinua\ \hat{}\ lineaActual.longitud \geq 10$

$b\ (q_i\ ,\ q_j\ ,I_n,\ A_1,\ A_M) \equiv a(q_i\ ,\ q_j\ ,I_n,\ A_1,\ A_M)\hat{}\ linea\ Anterior.id<lineaActual.id\ \hat{}$

$$\wedge\ lineaActual.concectada(lineaAnterior)$$

$$c\ (q_i\ ,\ q_j\ ,I_n,\ A_I\ ,\ A_M) \equiv \neg\ lineaActual.esContinua \wedge\ lineaAnterior.id > linActualId \wedge$$

$$\wedge\ lineaActual.concectada(lineaAnterior)$$

$$d\ (q_i\ ,\ q_j\ ,I_n,\ A_I\ ,\ A_M) \equiv \neg\ lineaActual.esContinua \wedge\ lineaAnterior.id < lineaActualId \wedge$$

$$\wedge\ lineaActual.concectada(lineaAnterior)$$

$$e\ (q_i\ ,\ q_j\ ,I_n,\ A_I\ ,\ A_M) \equiv lineaActual.concectada(lineaPrimera)$$

Para completar el proceso hay que comprobar la coincidencia de segmentos mencionada en ambas formas combinadas.

3.1.4.5 Detección de componentes

En la detección de componentes se sigue con la idea de realizar un autómata programado [Flasiński, 2002], [Flasiński, 1995] que en un recorrido de los segmentos horizontales y verticales existentes pueda encontrar una secuencia de los mismos que forme una estructura coincidente con la de un componente. En este caso los requisitos son que existan doce líneas conectadas conforme a la Ilustración III-10, condición suficiente para identificar un componente. Además se exigirá que los siete últimos lados encontrados tengan una longitud inferior de 40 pixeles.

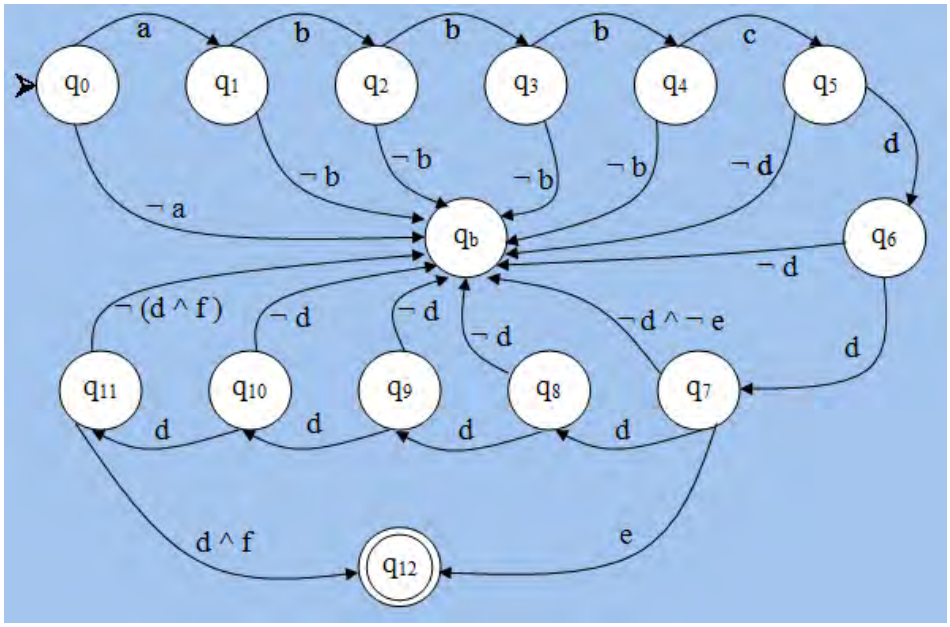


Ilustración III-19: Autómata para la detección de componentes.

Las definiciones de las expresiones lógicas son:

$a\ (q_i\ ,\ q_j\ ,I_n,\ A_I,\ A_M) \equiv lineaActual.esContinua\ \wedge\ lineaActual.longitud \geq 10$

$b(q_i\ ,\ q_j\ ,I_n,\ A_I,\ A_M) \equiv a(q_i\ ,\ q_j\ ,I_n,\ A_I,\ A_M)\ \wedge\ linea\ Anterior.id<linActual.id$

$\wedge\ lineaActual.concectada(lineaAnterior)$

$c\ (q_i\ ,\ q_j\ ,I_n,\ A_I,\ A_M) \equiv linActual.esContinua\ \wedge\ lineaAnterior.id>linActualId\ \wedge$

$\wedge\ linActual.concectada(lineaAnterior)$

$d(q_i\ ,\ q_j\ ,I_n,\ A_I,\ A_M) \equiv linActual<45\ \wedge\ linActual.esContinua\ \wedge$

$\wedge\ lineaAnterior.id>linActualId\ \wedge\ linActual.concectada(lineaAnterior)$

$e(q_i\ ,\ q_j\ ,I_n,\ A_I,\ A_M) \equiv linActual<45\ \wedge\ linActual.esContinua\ \wedge$

$\wedge\ lineaAnterior.id>linActualId\ \wedge\ linActual.concectada(primeraLinea)$

$f\ (q_i\ ,\ q_j\ ,I_n,\ A_I,\ A_M) \equiv lineaActual.concectada(lineaPrimera)$

Aunque en los requisitos de la investigación el reconocimiento de paquetes se restringía a la forma mostrada en la Ilustración III-10. El autómata representado en la Ilustración III-19 también reconoce la siguiente forma asociada a componentes:

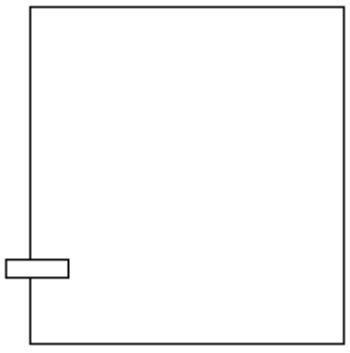


Ilustración III-20: Forma alternativa de representación de componentes

3.1.5 **Detección de segmentos oblicuos**

La detección de los segmentos oblicuos presentes en la imagen se realiza mediante una técnica de detección de líneas, el método de RANSAC [Fischler y Bolles, 1981]. Las discontinuidades de más de cinco píxeles determinan los extremos de los diferentes segmentos oblicuos presentes en la línea. La densidad de píxeles en cada segmento

candidato determina su identificación y el que sea clasificado como continuo o discontinuo. La información almacenada de cada segmento oblicuo es similar a la de segmentos verticales y horizontales incluyendo las conexiones entre ellos.

Cabe mencionar que para alcanzar mayor eficiencia los píxeles ya incluidos en algún segmento identificado no intervienen en el cálculo de ningún otro segmento.

3.1.6 **Detección de enlaces**

Los segmentos horizontales y verticales que no son lados de ninguna de las figuras reconocidas por autómatas son candidatos a pertenecer a enlaces, junto con los oblicuos. Los extremos de las secuencias de estos segmentos determinan los enlaces entre los elementos del diagrama próximos a ellos. La clasificación de los enlaces dependerá del símbolo de la cabeza de la relación. La detección de estas cabezas se aborda en el siguiente epígrafe 3.1.7.

Mención aparte reciben las relaciones de jerarquía compuestas por una sucesión de líneas que no siempre se conectan por los extremos. En la Ilustración III-21 se contempla un ejemplo de la situación descrita.

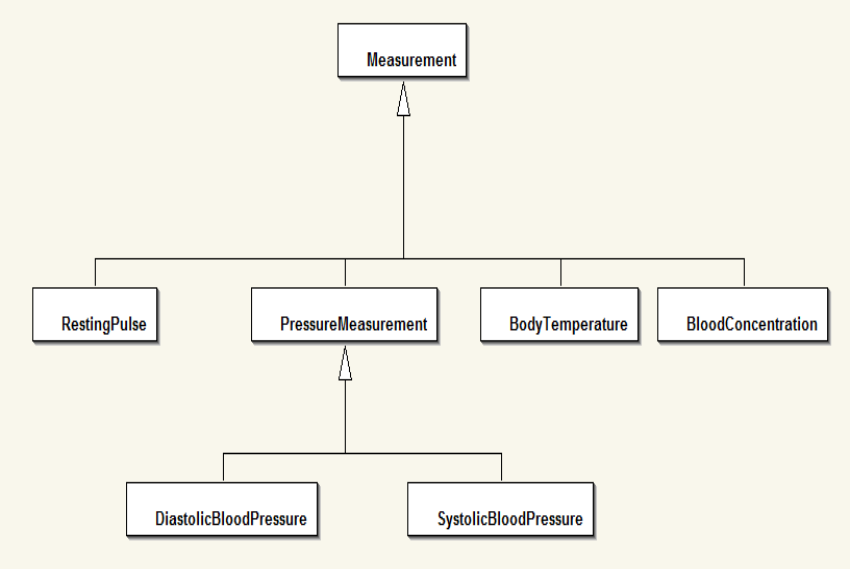


Ilustración III-21: Ejemplo de relación compleja de jerarquía.

Para ello se ha diseñado el siguiente algoritmo, explicado mediante una simulación de cómo detectaría la herencia del diagrama UML de la Ilustración III-21:

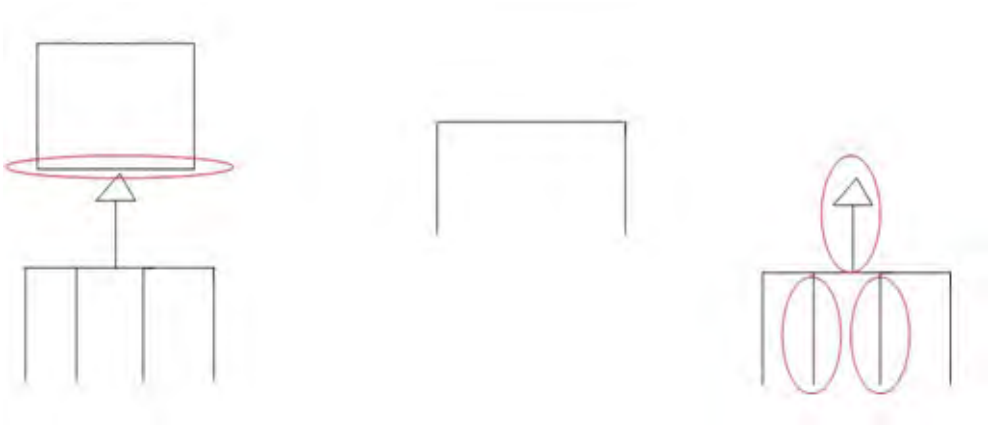


Ilustración III-22: Proceso de detección de enlaces complejos.

Los pasos son los siguientes:

- Se ha detectado una cabeza de relación en lado inferior de una forma y no se ha detectado un enlace simple asociado.
- Se detecta la forma central de la Ilustración III-22
- Se completan los enlaces a partir de ella

3.1.7 **Detección de formas por circularidad**

La circularidad (ver epígrafe 2.3.6) permite detectar formas compuestas por segmentos difíciles de identificar por su escasa longitud o carentes de ellos.

La representación de la interface en los diagramas UML es una de las formas que se identifican con esta técnica. El valor teórico de la circularidad para esta figura es 12.56 y el intervalo práctico de valores admitidos para el reconocimiento, obtenido tras experimentar con 50 ejemplos presentes en diagramas UML, es: [11.42, 12.78]

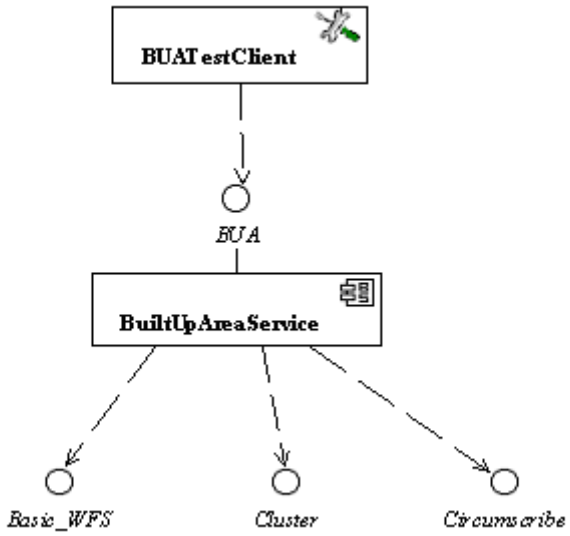


Ilustración III-23: Ejemplo de imagen con interfaces

En la Ilustración III-23 se muestran cuatro interfaces. Sus valores de circularidad de arriba abajo y de izquierda a derecha son: 12.5772 12.7217 12.7217 12.7217

Otra de las formas reconocidas calculando su circularidad es la que representa relaciones n-arias. El valor teórico de esta forma es 16 y el intervalo de valores experimentales obtenido tras procesar 50 ejemplos es: [14.96, 16.68]

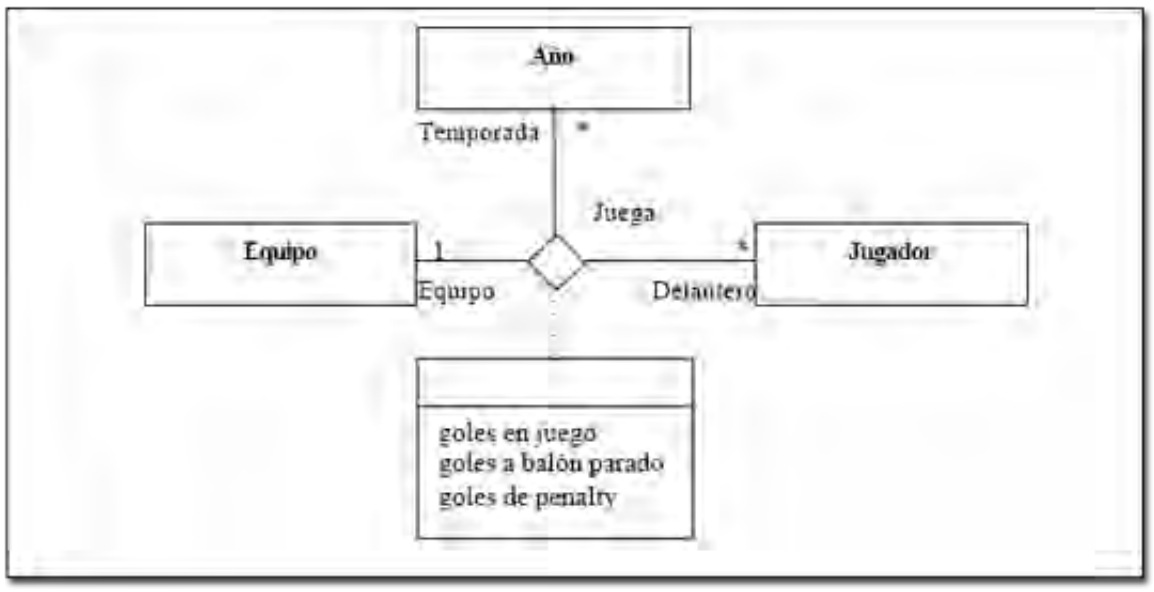


Ilustración III-24 Ejemplo de imagen con relación n-aria.

El resto de formas corresponde a cabezas de relaciones, o bien llamándolas de forma más descriptiva, puntas de flechas. El proceso a seguir comienza con la delimitación y aumento de tamaño de las zonas de la imagen donde están los extremos de las relaciones identificadas según el epígrafe 3.1.6. En la siguiente Ilustración III-25 se observan ejemplos de capturas de tamaño 50 x 50.

Ilustración III-25: Capturas de cabeza de relaciones

El aumento de tamaño permite obtener valores de circularidad más exactos y la acotación de la zona mejora ostensiblemente la eficiencia.

Clasificando manualmente en torno a 50 figuras de cada tipo (composición, agregación, asociación y jerarquía) se han obtenido intervalos de aceptación no deterministas por existencia de solapamientos. Por tanto, ha sido necesario realizar la discriminación atendiendo a otros criterios:

- Numero de valores de circularidad
  La jerarquía y la agregación devuelven dos valores (uno de la forma externa y otra de la interna)

- Porcentaje de píxeles negros en relación al perímetro
  Destaca la composición frente a la asociación en esta medida

- Coincidencias de los píxeles que son extremos
  Tomando 4 píxeles del perímetro (el situado más arriba, más abajo, más a la izquierda y más a la derecha de la figura) estos pueden estar repetidos (o ser muy próximos). En las jerarquías deben de coincidir dos de los 4 píxeles extremos y en

las agregaciones no debe haber ninguna coincidencia, ya que cada uno de los 4 píxeles extremos se corresponde con un vértice de la figura.

3.1.8 **Experimentación y resultados**

La experimentación se ha llevado a cabo sobre un conjunto de 117 imágenes obtenidas de la web. Para las cabezas de relaciones se ha experimentado sobre un subconjunto de 40 imágenes y en el caso de relaciones n arias sobre uno de 20. Todas las imágenes cumplen los criterios establecidos en las restricciones de la investigación en cuanto a resoluciones, formas reconocibles y demás aspectos.

Se ha verificado que no hay imágenes repetidas y se ha minimizado la evaluación del mismo tipo de figura en la misma imagen. Esta última consideración se debe al hecho habitual de utilizar opciones de pegado en el diseño de los diagramas. Por ejemplo, en la

Ilustración III-23 observando los valores de las mediciones de la circularidad en los interfaces se puede concluir que al menos tres de ellos son idénticos. Es decir, que hay muchas posibilidades de tener evaluaciones repetidas de cada una de las formas de una misma imagen, y por tanto, se podrían desvirtual los resultados. Bajo esta premisa se ha realizado una selección manual al azar de los elementos a identificar en cada imagen.

También se ha evitado trabajar con imágenes diseñadas ad hoc con el propósito de realizar validaciones automáticas. Aunque es evidente que las evaluaciones serían más rápidas al tener la información a priori de los modelos representados, la evaluación estaría afectada por trabajar con imágenes preparadas cuya procedencia no es la web.

El éxito de la detección se ha verificado mediante validación humana. Se muestran a continuación los resultados:

Forma	Nº de ejemplares	Porcentaje de acierto
Segmentos horizontales y verticales	100	100%
Segmentos oblicuos	100	83%
Clases	100	100%
Paquetes	50	98%
Notas	50	94%
Clases parametrizadas	50	78%
Relaciones n-arias	20	85%
Interfaces	50	98%
Conexión Enlaces	50	84%
Dependencia	48	85%
Asociación bidireccional	50	86%
Componentes	50	82%
Jerarquía	47	80,85%
Agregación	46	80,43%
Composición	46	84,78%

Tabla III-2: Porcentaje de acierto en la detección de formas

3.2 Filtrado de imágenes de diagramas UML de procedencia web

La Reutilización es una rama de la Ingeniería de Software que estudia métodos para aprovechar el esfuerzo realizado en desarrollos de software anteriores. Se trata de utilizar artefactos software existentes en la construcción de nuevos productos software. Estos artefactos pueden ser: código fuente, diseños, módulos, especificaciones, arquitecturas, pruebas, etc. Por lo tanto en este contexto es importante la utilización de técnicas de recuperación de información para hacer posible su reutilización.

La reutilización en fases de alto nivel como el diseño resulta más provechosa que en otras etapas inferiores como la implementación. Esto se debe a que las fases de alto nivel requieren una mayor inversión de tiempo en el desarrollo software. Además a partir de ellas se pueden reutilizar ramas completas del ciclo de vida del software.

La documentación correspondiente a estas etapas de alto nivel son por tanto muy valoradas en el ámbito de la recuperación. En particular los diagramas UML por su estructura y contenido resultan muy apropiados para este tipo de tareas. Sin embargo, es necesario disponer de una amplia colección de este tipo de documentación con el fin de seleccionar los diagramas más afines al proyecto software que se desea desarrollar. En

este sentido, se plantea la idea de construir estos repositorios recopilando de la Web cualquier documento que encaje con estos tipos de diagramas.

Para recuperar diagramas UML de la Web se utilizan motores de búsqueda. Surge entonces el problema de comprobar si todos los resultados que ofrecen los buscadores web son pertinentes con las demandas de información realizadas. Es decir, para cada resultado es necesario conocer si realmente se trata de un diagrama UML antes de que sea incluido en el repositorio.

Este fase de la investigación se centra en recuperar de la Web diagramas UML en formato imagen. Para ello se ha utilizado el buscador de imágenes Google Image. Las consultas realizadas demandan al motor de búsqueda de imágenes diagramas UML de diversas temáticas. Una vez obtenidas las imágenes se desea identificar automáticamente aquellas que realmente son diagramas UML.

Para desarrollar esta fase se establece una metodología, y se diseñará una experimentación de la que se expondrán y discutirán los resultados obtenidos

3.2.1 Metodología para la identificación automática de imágenes de diagramas UML

Con el propósito de discriminar si una imagen es o no un diagrama UML se utiliza aprendizaje supervisado.

Los pasos seguidos en la metodología aplicada en la identificación de las imágenes que representan diagramas son:

Etapa 1: Construcción de modelos para la clasificación de imágenes

- **Obtención de ejemplos reales:** Por medio de consultas web se obtienen imágenes.
- **Clasificación manual:** Se etiquetan las imágenes manualmente.
- **Identificación de las características**: Se identifican características representativas y se extraen sus valores de cada imagen.
- **Construcción de modelos:** Se crean modelos de clasificación aplicando aprendizaje supervisado.

Etapa 2: Validación de los modelos construidos

En esta etapa se aplican los modelos construidos a conjuntos de instancias etiquetadas que no han intervenido en el proceso de aprendizaje. Para ello se extrae de las imágenes, pertenecientes a conjuntos de test, los valores de las características que se seleccionaron en el entrenamiento. Posteriormente se clasifican con los modelos construidos.

Etapa 3: Resultados de evaluación

A partir de las imágenes clasificadas pertenecientes a los conjuntos de test se obtienen mediciones del porcentaje de acierto, la precisión y el recall, que nos permite valorar la exactitud de los modelos. Las definiciones de estas medidas en función de los números de instancias clasificadas como *TP* (verdaderos positivos), *FP* (falsos positivos), *TN* (verdaderos negativos) y *FN* (falsos negativos) son:

[8] $porcentaje\ de\ acierto\ =\ (TP+TN)/(TP+TN+FP+FN)$

[9] $precisión\ =\ TP/(TP+FP)$

[10] $recall\ =\ TP/(TP+FN)$

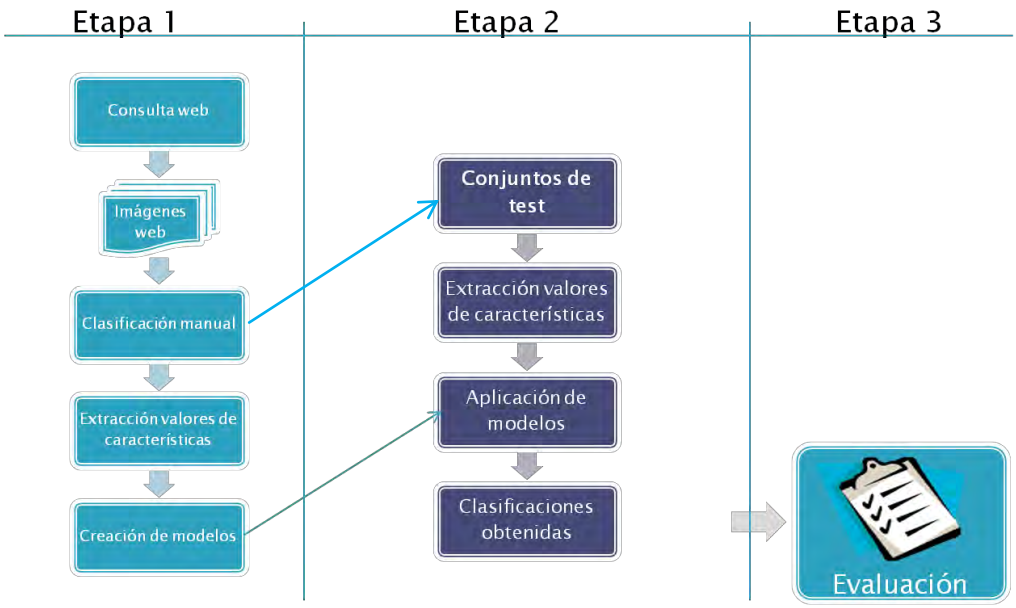


Ilustración III-26: Descripción de etapas y apartados del filtrado de imágenes

3.2.2 **Desarrollo**

Los puntos tratados en la metodología con el propósito de identificar imágenes de diagramas UML se desarrollan en este apartado.

3.2.2.1 Obtención de ejemplos reales

La muestra de estudio de imágenes de la Web que se ha analizado se ha obtenido a partir de consultas que demandaban a Google Image diagramas de UML de distintas temática. Se han descargado como respuesta a estas consultas 18899 imágenes que han sido clasificadas manualmente por expertos en UML según los criterios de clasificación recogidos en el Anexo A.

El número de imágenes clasificadas en cada categoría se muestra en la Tabla III-3:

CÓDIGO	DESCRIPCIÓN	Nº de imágenes
1	Representación de diagrama UML	794
2	Representación de diagrama no UML	6703
3	Imagen con información desestructurada	11402

Tabla III-3: Número de imágenes clasificadas manualmente según su categoría

Se ha procedido a una revisión cruzada (entre los expertos) de cada una de las imágenes clasificadas con el fin de minimizar la subjetividad y los errores inherentes a las etiquetaciones manuales.

3.2.2.2 Identificación de las características

En este punto se indican las características que se tienen en cuenta para clasificar imágenes. Las características que se consideran son:

Histograma de escala de grises

Las imágenes que representan diagramas UML suelen tener un número muy limitado de tonos diferentes de gris. Esto es debido a que los ingenieros al diseñarlos con herramientas gráficas emplean en general un conjunto de tonalidades reducido en pro de simplificar el resultado estético. El número de tonos de gris presentes en una imagen se puede obtener a partir del histograma de escala de grises de la imagen. Si la imagen viene representada por el modelo de color RGB el histograma de grises se puede obtener aplicando a cada uno de los píxeles la ecuación [7], redondeando posteriormente los

valores obtenidos al entero más próximo. En la Ilustración III-27 se muestra un ejemplo típico:

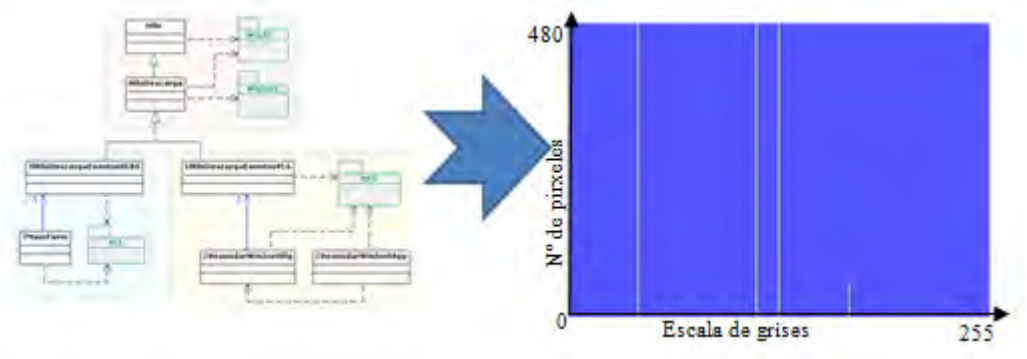


Ilustración III-27: Histograma de grises diagrama[8]

En este caso se contempla que los píxeles de la imagen se concentran, según sus niveles de gris, en cuatro tonos.

No obstante, esta característica no es por si sola completamente discriminante. Existen imágenes que no representan diagramas, incluso las fotográficas, pueden presentar un número reducido de tonos de gris. La siguiente ilustración es una muestra de ello.

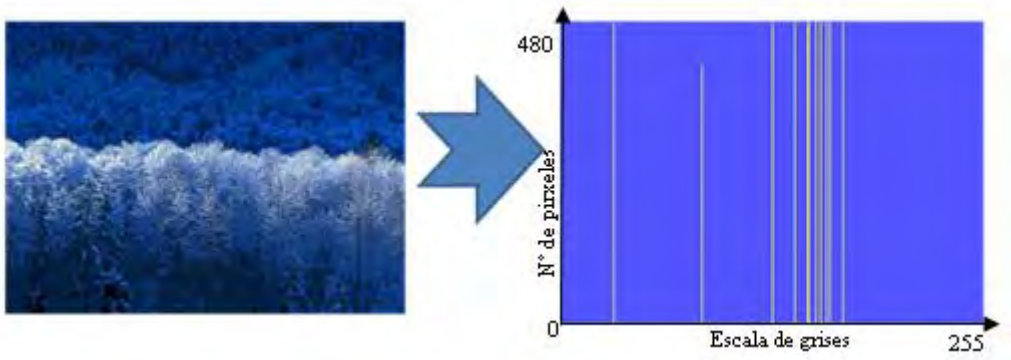


Ilustración III-28: Histograma de grises de imagen fotográfica[9]

[8] Imagen de diagrama tomada de http://articulo.mercadolibre.com.mx/MLM-18487899-cursos-completos-manuales-de-c-ejemplos-envio-gratis-_JM (Crawled on 16/04/2008)

[9] Imagen fotográfica de Thad Allender tomada de http://thadallender.photoshelter.com/image/I0000Xlltl1LV9h0 (Crawled on 14/09/2009)

Por otra parte, otras que si representan diagramas pueden abarcar todo el abanico de tonos. Un ejemplo de este tipo de imagen se mostro en la Ilustración III-4 y su histograma en la Ilustración III-5.

En ese caso se observa como la imagen distribuye sus píxeles entre todos los valores de gris de la escala. Esto se debe a que posee degradados o difuminaciones incorporadas como elementos decorativos.

Aplicar por tanto esta única característica llevaría a descartar imágenes que representan diagramas UML y a seleccionar otras carentes de información gráfica sobre modelos.

Histograma de escala de tonos del modelo HSI

El histograma basado en tonalidad de color (canal *H* del modelo *HSI*) puede considerarse una característica discriminante debido a que los diseñadores de diagramas tienden a la uniformidad utilizando un reducido número de valores colorimétricos. A pesar de ello, hay excepciones en ambos sentidos. La siguiente ilustración muestra un diagrama UML en formato imagen rica en tonalidades cromáticas.

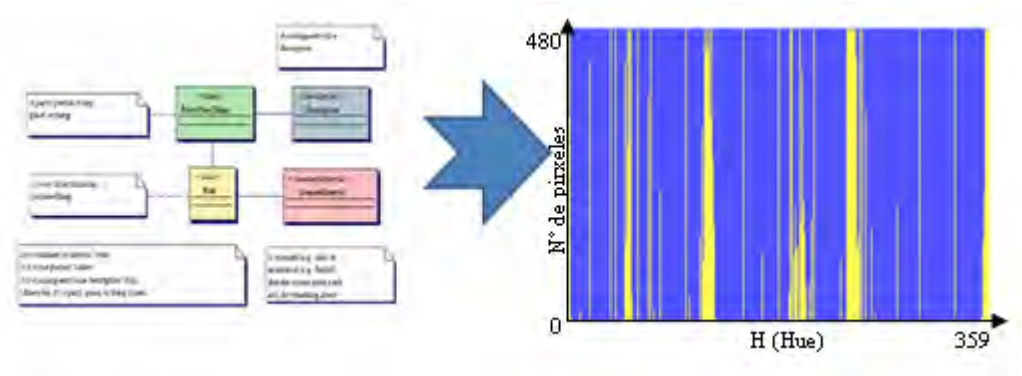


Ilustración III-29: Histograma de escala de color (H) diagramaUML[10]

[10] Imagen de diagrama de Jose Alfredo Ortiz Romero tomada de https://sites.google.com/site/analisisdiazirving/unidad-ii/resumen-3

Mientras que la siguiente ilustración presenta el histograma de color de la imagen (

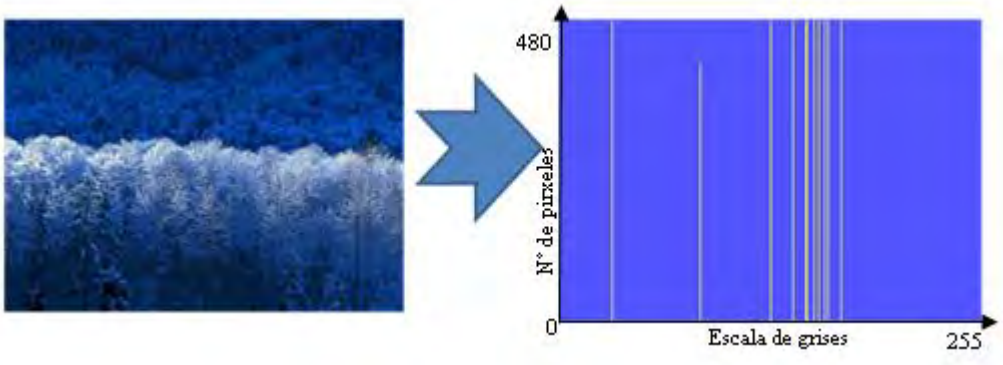


Ilustración III-28), poniendo de manifiesto el reducido número de tonos presentes en algunas imágenes paisajísticas.

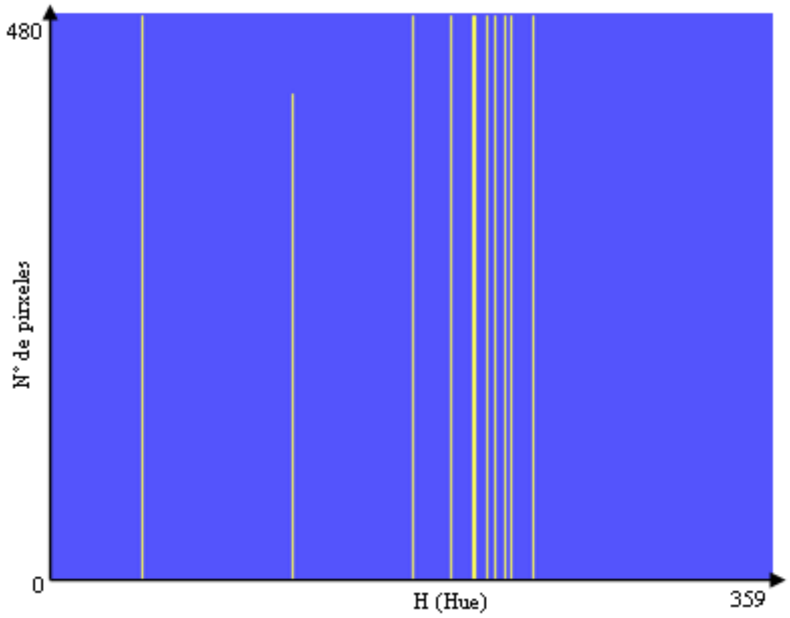


Ilustración III-30: Histograma de color de imagen fotográfica

Formas detectadas en las imágenes

Se hace necesario introducir otras características con potencial discriminante a la vista de lo anteriormente expuesto. La información extraída en el proceso de detección de formas, como por ejemplo el número de rectángulos y el número de líneas, tiene un indudable aporte en la discriminación de diagramas UML. En el siguiente par de ilustraciones se muestra un ejemplo.

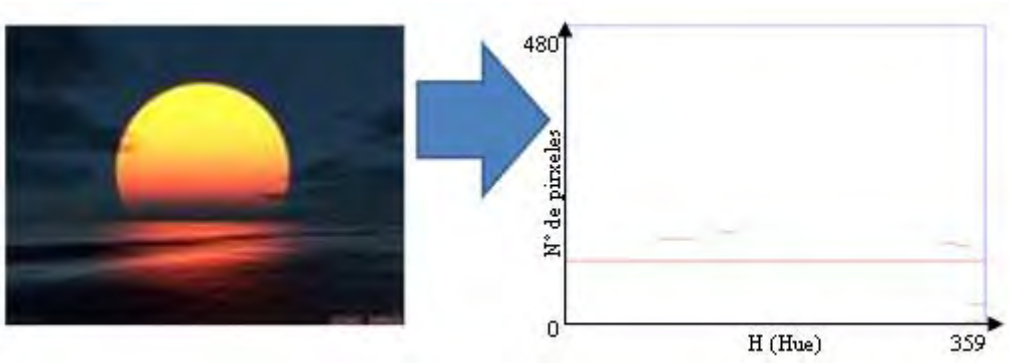


Ilustración III-31: Rectas fotografía Ocaso[11]

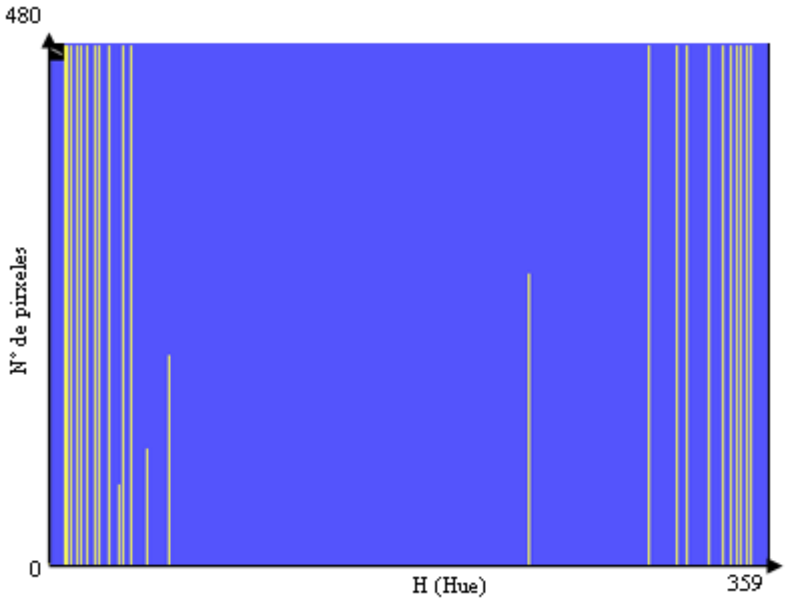


Ilustración III-32: Histograma de colores de fotografía Ocaso.

Se puede observar en las ilustraciones (Ilustración II-31, Ilustración III-32**)** como la búsqueda de determinadas formas geométricas simples, en este caso líneas horizontales, permiten discernir que no se trata de un diagrama UML, a pesar de que su histograma de tono de color podría indicar lo contrario. El reducido número obtenido de estas líneas es incompatible con la representación de un diagrama.

[11] Imagen fotográfica de Peter Kleiner tomada de https://www.flickr.com/groups/flickr_aid/ (Crawled on 30/05/2012)

Finalmente, tras pruebas preliminares, el conjunto de características o atributos identificado por su valor discriminante para la clasificación de imágenes según representen o no modelos UML (de los caso de estudio) es el formado por:

- Número de tonos presentes en el histograma de escala de color
- Número de tonos presentes en el histograma de escala de grises
- Número de líneas detectadas
- Número de líneas detectadas de longitud mayor o igual a 30 píxeles
- Número de líneas detectadas en posición horizontal (longitud ≥ 30 píxeles)
- Número de líneas detectadas en posición vertical (longitud ≥ 30 píxeles)
- Número de rectángulos detectados
- Número de rectángulos principales detectados (no están incluidos en otros)

3.2.2.3 Generación automática de reglas para la clasificación de imágenes

La detección de las imágenes, que representan diagramas UML, se realiza por un clasificador automático. Este clasificador se ha construido utilizando técnicas de minería de datos de aprendizaje supervisado. Se han tomado como ejemplos de aprendizaje las 18899 imágenes clasificadas por los expertos en UML considerando dos categorías diagramas UML (código 1) y otras imágenes (códigos 2 y 3).

A partir de este conjunto clasificado de imágenes se pretende extraer un patrón o modelo de comportamiento que, posteriormente se pueda extrapolar a las nuevas imágenes que lleguen al sistema y que permita la clasificación de las mismas como diagramas UML u otras imágenes. Con este propósito se ha experimentado combinando clasificadores homogéneos mediante la técnica Bagging, tomando como base el algoritmo de inducción de reglas PART [Frank y Witten, 1998], por las ventajas mencionadas en el epígrafe 2.5.2.

Estas ventajas permiten minimizar el impacto de posibles errores en la clasificación manual de las imágenes. Además, de incorporar fácilmente reglas deducidas por expertos.

Se ha utilizado la herramienta de análisis de datos Weka [Witten y Frank, 2000] [Witten y Frank, 2005] en la que están implementados los anteriores algoritmos.

Las instancias o ejemplos de aprendizaje se han construido a partir de la relación de atributos del punto anterior (histogramas de escalas de grises y color, del número de diferentes tipos de líneas y formas rectangulares obtenidas mediante técnicas de

procesado de imágenes). La combinación de estos atributos junto al atributo de clase proporcionado por los expertos proporciona una instancia por cada una de las imágenes.

3.2.3 **Experimentación y resultados**

Se han realizado 10 experimentos. En cada uno de ellos han intervenido las 794 instancias positivas (asociadas a diagramas UML) y otras tantas instancias negativas (correspondientes a otros tipos de imágenes) tomadas al azar sin reemplazamiento del total de instancias negativas.

En los experimentos se ha utilizado la técnica de conjunto de clasificadores homogéneos Bagging tomando como base el algoritmo PART. Se ha mantenido la configuración de parámetros que la herramienta Weka tiene por defecto.

Los resultados obtenidos tras evaluar, mediante evaluación cruzada con 10 divisiones estratificadas, los clasificadores obtenidos se muestran en las siguientes tablas:

Experimento	% Aciertos	%Fallos
1	94,2695 %	5,7305%
2	94,2065 %	5,7935%
3	94,3955 %	5,6045%
4	95,0882 %	4,9118%
5	95,2771 %	4,7229%
6	94,3955 %	5,6045%
7	93,0101 %	6,9899%
8	94,7733 %	5,2267%
9	93,8287 %	6,1713%
10	94,2065 %	5,7935%
Promedio	94,3451 %	5,6549%

Tabla III-4: Porcentajes de acierto y error en la clasificación de imágenes

Experimento 1	Diagrama UML	Otras imágenes
Precisión	0.956	0.93
Recall	0.928	0.957

Experimento 2	Diagrama UML	Otras imágenes
Precisión	0.95	0.934
Recall	0.933	0.951

Experimento 3	Diagrama UML	Otras imágenes
Precisión	0.945	0.943
Recall	0.943	0.945

Experimento 4	Diagrama UML	Otras imágenes
Precisión	0.965	0.938
Recall	0.936	0.966

Experimento 5	Diagrama UML	Otras imágenes
Precisión	0.963	0.943
Recall	0.942	0.963

Experimento 6	Diagrama UML	Otras imágenes
Precisión	0.956	0.933
Recall	0.931	0.957

Experimento 7	Diagrama UML	Otras imágenes
Precisión	0.936	0.924
Recall	0.923	0.937

Experimento 8	Diagrama UML	Otras imágenes
Precisión	0.955	0.941
Recall	0.94	0.956

Experimento 9	Diagrama UML	Otras imágenes
Precisión	0.952	0.925
Recall	0.923	0.953

Experimento 10	Diagrama UML	Otras imágenes
Precisión	0.953	0.931
Recall	0.929	0.955

Promedios	Diagrama UML	Otras imágenes
Precisión	0.953	0.934
Recall	0.933	0.954

Tabla III-5: Resultados de precisión y recall de los experimentos

Ejemplos de clasificación:

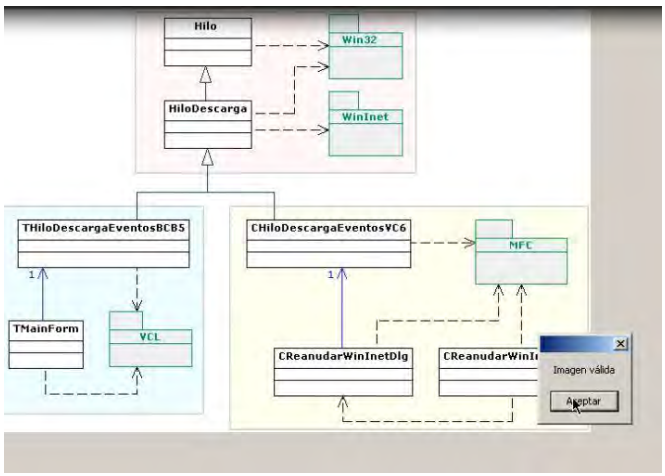


Ilustración III-33: Imagen clasificada como diagrama UML

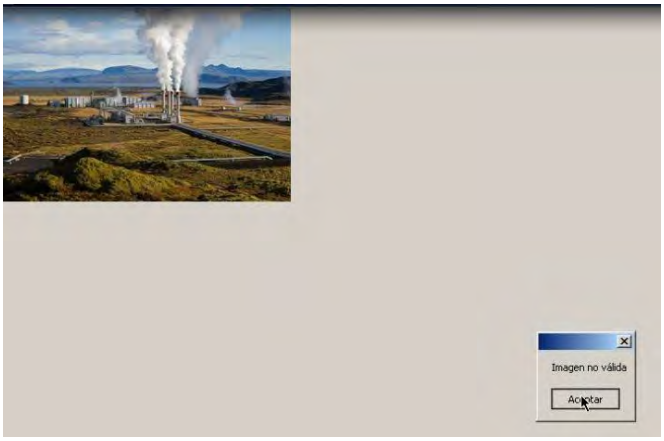


Ilustración III-34: Imagen clasificada como no diagrama UML[12]

3.3 Reconocimiento del texto presente en imágenes UML

Una vez reconocida la información estructural de la representación de diagramas UML el siguiente paso consiste en reconocer el texto asociado a cada forma. El principal problema radica en la baja resolución que pueden presentar las imágenes. Si estas proceden de la Web la resolución estará entre 72 y 96 dpi, muy lejos de las recomendaciones de uso de las herramientas diseñadas para este fin, los OCR, que suelen estar en 300 dpi.

3.3.1 **Metodología para el uso de OCR en diagramas UML**

En el Anexo A se presenta un estudio previo (Computer Vision Group - Universidad Politécnica, 2008), enmarcado dentro del proyecto UML Models (desarrollado por el grupo de investigación Knowledge Reuse), comparando diferentes softwares OCR aplicados a imágenes Web. Además, incluye un método (Ilustración III-35) que permite optimizar la aplicación MODI del paquete de Microsoft Office para ese fin. El interés de esta aplicación consiste en que su uso no requiere de una inversión adicional por ser habitual disponer de la licencia pertinente y en que su eficacia es superior al del resto de aplicaciones OCR libres, según el estudio mencionado.

[12] Imagen fotográfica original de Gretar Ívarsson tomada de http://en.wikipedia.org/wiki/Template:POTD/2007-11-25 (Crawled on 07/02/2008)

El método que se propuso se inicia con una conversión de la imagen a escala de grises, después se aumenta la resolución de la imagen hasta que el OCR pueda procesar la imagen proporcionando información del tipo de letra utilizado en cada área. Según el tipo de grafía detectado se aplicara interpolaciones diferentes y se procederá al reconocimiento final del texto aplicando nuevamente la herramienta MODI.

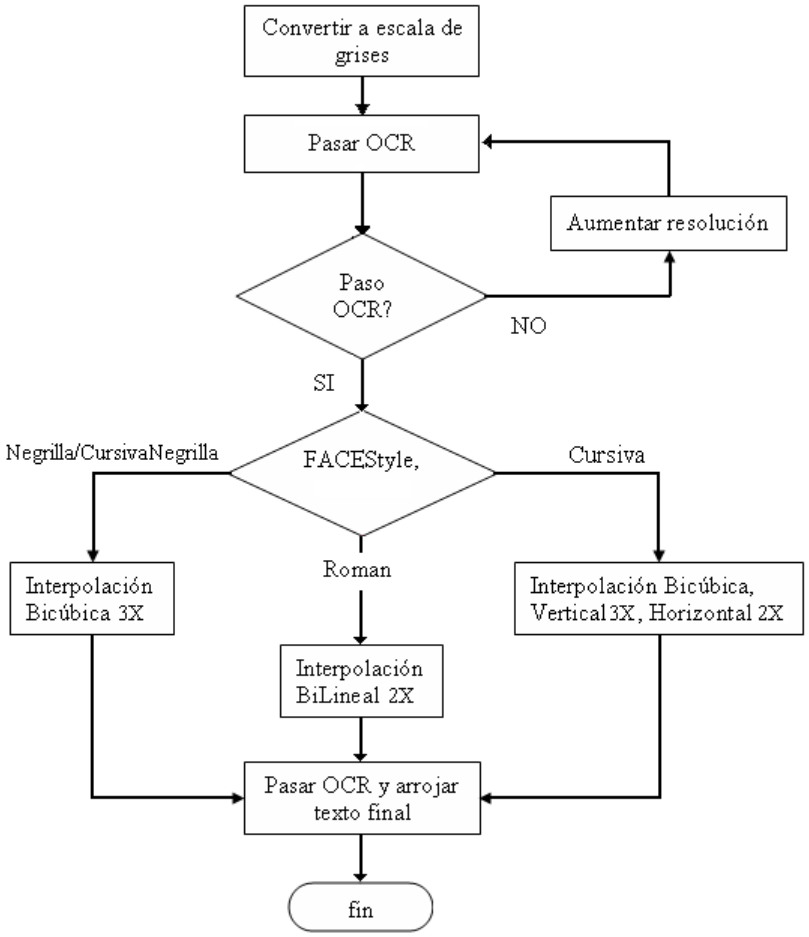


Ilustración III-35: Diagrama de flujo, algoritmo para reconocimiento de texto

Se han investigado las distintas formas de obtener los mejores resultados con la manipulación previa de la imagen. La solución final a aplicar consiste en realizar una primera pasada del OCR sobre la imagen para obtener todas las palabras existentes en la imagen. Estas se almacenan en una estructura de datos junto con información relativa a su fuente de letra y su estilo, así como su posición en la imagen gracias a las coordenadas proporcionadas por el OCR. El siguiente paso a realizar es el de generar una imagen nueva que contenga solamente la información útil relativa al texto de la imagen, así como las modificaciones que se consideren necesarias sobre las palabras para mejorar su lectura. En este caso se diferenciarán tres tipos posibles de letra:

Palabra en negrita: En este caso la palabra a introducir en la imagen nueva será dicha palabra remuestreada y ampliada por 3 tanto en ancho como en alto.

Palabra en cursiva: En este caso la palabra a introducir en la imagen nueva será dicha palabra remuestreada y ampliada por 3 en ancho y por 2 en alto.

Palabra en roman: En este caso la palabra a introducir en la imagen nueva será dicha palabra remuestreada y ampliada por 2 tanto en ancho como en alto.

Las palabras se irán poniendo una debajo de la otra. Una vez se ha generado la nueva imagen con las transformaciones correspondientes de cada palabra se pasará de nuevo el OCR, y se actualizará la información obtenida para cada palabra. La información a actualizar será únicamente el texto reconocido, y no las coordenadas, ya que al haber pasado el OCR sobre una imagen totalmente distinta las coordenadas no concordarían con el resto de elementos del diagrama UML.

Una vez actualizada la información de cada palabra se procede a relacionar cada palabra dentro de cada elemento (clase, nota, componente, clase parametrizada,…). Esto se realiza comprobando que la palabra está dentro de los límites de un elemento y no sobresale del mismo, dándole un margen de error de dos pixeles por cada lado.

El siguiente proceso a realizar es la concatenación de palabras. El OCR va buscando palabras sueltas, pero es evidente que, para el caso de diagramas UML, estas palabras carezcan de significado por si solas ya que pueden pertenecer a un conjunto de palabras que aportan una información específica, como puede ser el caso de un atributo (compuesto por la tupla: *nombre de la variable* + *tipo de la variable*). Para ello, las palabras que pertenezcan a un mismo elemento y se encuentren en una misma línea serán almacenadas juntas, de forma que se tenga agrupada por líneas la información relativa a cada elemento existente en el diagrama.

El último paso a realizar es el de mejorar la información obtenida de cada línea. Para ello se hace un estudio de cada línea, de forma que dependiendo del elemento, se obtenga que tipo de información está aportando cada línea. Para ello se realiza un análisis del texto obtenido de acuerdo al elemento al que pertenezca. La información posible es la siguiente:

- Clase

- Atributo: Compuesto por la tupla *nombreAtributo: tipoAtributo.* Pueden existir de 0 a n atributos por clase.
- Método: Compuesto por la siguiente estructura *nombreMetodo(arg1: tipo, arg2: tipo,...)*. Pueden existir de 0 a n métodos por clase.
- NombreClase: Compuesto por una sola palabra que empieza por mayúscula. Solo puede existir un nombre de clase por clase.

- Nota
  - Texto: Simplemente contiene anotaciones relacionadas al diagrama, por lo que no se realiza ningún tratamiento especial.
- Paquete
  - Atributo: Compuesto por la tupla *nombreAtributo: tipoAtributo.* Pueden existir de 0 a n atributos por paquete.
  - Método: Compuesto por la siguiente estructura nombreMetodo(arg1: tipo, arg2: tipo,…). Pueden existir de 0 a n métodos por paquete.
  - NombrePaquete: Compuesto por una sola palabra que empieza por mayúscula. Solo puede existir un nombre de paquete por paquete.
- Componente
  - Atributo: Compuesto por la tupla *nombreAtributo:tipoAtributo.* Pueden existir de 0 a n atributos por componente.
  - Método: Compuesto por la siguiente estructura *nombreMetodo(arg1: tipo, arg2: tipo,...)*. Pueden existir de 0 a n métodos por componente.
  - NombreComponente: Compuesto por una sola palabra que empieza por mayúscula. Solo puede existir un nombre de clase por componente.
- Clase Parametrizada
  - Atributo: Compuesto por la tupla *nombreAtributo:tipoAtributo.* Pueden existir de 0 a n atributos por clase parametrizada.
  - Método: Compuesto por la siguiente estructura *nombreMetodo(arg1: tipo, arg2: tipo,...)*. Pueden existir de 0 a n métodos por clase parametrizada.
  - NombreClaseParametrizada: Compuesto por una sola palabra que empieza por mayúscula. Solo puede existir un nombre de clase por clase parametrizada.
  - TipoParametrizacion: Contiene la información relativa a la parametrización que posee la clase en cuestión

3.3.2 **Resumen de mejoras en la metodología**

Se han practicado cambios en la metodología orientados principalmente a mejorar la eficiencia.

- Automatización del proceso experimental permitiendo un aumento significativo en el nº de imágenes de muestra.
- Eliminación de elementos gráficos (líneas verticales) próximos al texto. Se aprovecha la detección de formas previa al reconocimiento de texto para eliminar los segmentos verticales identificados. En algunas imágenes se mejora sensiblemente los porcentajes de reconocimiento de texto.
- Reducción de la superficie de actuación del OCR. Actúa sobre el interior de las formas detectadas. Estas superficies son agrupadas en una misma imagen minimizando de esta manera el área de actuación del OCR.
- Aumento inicial de la resolución a 300 dpi. Se evita el bucle inicial (Ilustración III-36) que es excesivamente costoso.
- Creación de imágenes auxiliares por cada tipo de letra con la finalidad de unificar los textos con procesado común.

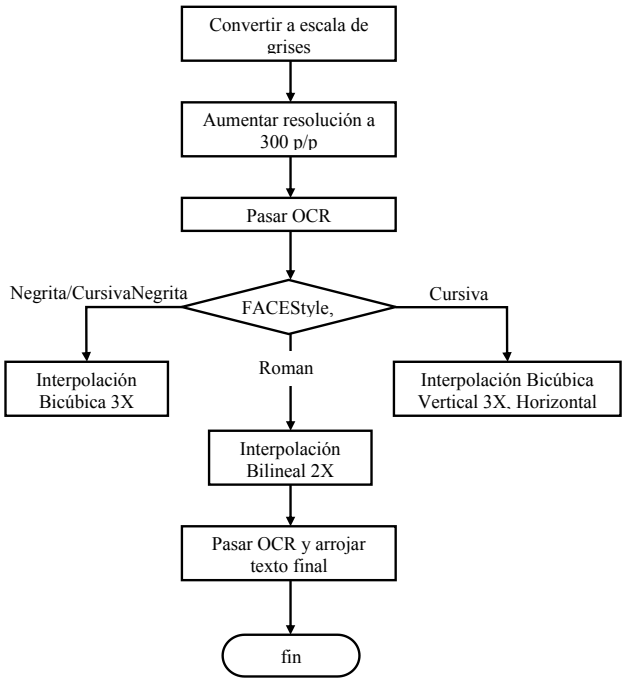


Ilustración III-36: Diagrama de flujo, algoritmo para la mejora del OCR

3.3.3 **Experimentación y resultados**

Se ha tomado, de forma automática de sus respectivas imágenes originales, el mismo juego de pruebas utilizado en el Anexo A , lo que permite realizar comparaciones de medidas de éxito de reconocimiento directamente. En la siguiente tabla se muestran los porcentajes de acierto en el reconocimiento de caracteres del OCR MODI. Los porcentajes corresponden a su aplicación directa, a su aplicación después de realizar el preporcesado de imágenes recomendado en el Anexo A, y a su aplicación después de efectuar el preprocesado con las modificaciones expuestas en este trabajo.

Aplicación directa OCR	**Con algoritmo de preprocesado**	**Con algoritmo de procesado modificado**
69 %	91%	94,6%

Tabla III-6: Porcentaje de acierto del OCR MODI según los distintos preprocesados

Finalmente se ha llevado a cabo un experimento para evaluar la eficiencia del reconocimiento de texto en comparación con la detección de elementos estructurales.

Se han procesado, en cada ejecución. 117 imágenes que representan modelos UML y se han calculado los tiempos medios a partir de las mediciones realizadas en 10 ejecuciones.

La siguiente tabla muestra los resultados obtenidos:

Actividad	**Tiempo**
Detección de formas	1, 43 sg
Reconocimiento Texto	13,89 sg
Total	**15,32 sg**

Tabla III-7: Tiempos medios por actividad

Capítulo IV: Discusión y Conclusiones

En este capítulo se recoge la discusión y conclusiones de esta investigación. Se ha estructurado en diferentes apartados que se corresponden con los bloques en los que se ha organizado este trabajo, junto con un extracto de las principales conclusiones.

4.1 Discusión sobre la detección de formas UML

La metodología propuesta en este bloque ha resultado adecuada. Y las decisiones tomadas en su desarrollo han repercutido de forma positiva tanto en la eficiencia de las detecciones como en los porcentajes de éxito de las mimas. Entre las decisiones más representativas destacan:

- La umbralización por contraste combinada de manera simultánea con detección de bordes. Estas técnicas junto a su aplicación combinada suponen un ahorro computacional y proporciona imágenes con bordes unitonales de exactamente un píxel de grosor. Además, se ha mostrado robusta aún en presencia de sombreados y otros tipos de manipulaciones ornamentales como degradados de color
- Distinguir entre segmentos horizontales, verticales y oblicuos ha permitido mejorar la eficiencia en su detección y alcanzar tasas del acierto del 100% en los dos primeros. La técnica propuesta para detectar los segmentos verticales y horizontales permite detectarlos con un simple recorrido. La detección de segmentos oblicuos también se ha visto favorecida por ello, ya que tras reconocer los segmentos horizontales y verticales se reduce el número de píxeles candidatos a formar parte de algún segmento oblicuo. Al estar formadas las figuras principales (clases, notas, componentes, elementos parametrizados y

componentes) por segmentos horizontales y verticales la detección correcta de ellos tiene aun mayor trascendencia.

Por otra parte, la técnica heurística empleada en la detección de segmentos oblicuos respeta el compromiso de eficiencia deseada y sus resultados incluso mejoran a los alcanzados en la literatura.

- Los autómatas programados han resultado esenciales en la detección precisa y eficiente de las formas principales. Su flexibilidad unida a la posibilidad de poder actuar simultáneamente en el reconocimiento de formas les hacen muy apropiados en la detección de figuras formadas por segmentos.
- La aplicación de medidas invariantes como la circularidad a áreas restringidas también ha resultado de gran utilidad en la detección de figuras a las que no se podía aplicar fácilmente las técnicas anteriores.

Entre las limitaciones se encuentran algunas descritas en trabajos referenciados en el estado del arte, como son los errores en la detección de relaciones por omisiones en la detección de segmentos oblicuos o de las cabezas de las relaciones, y la presencia de falsos positivos sobre todo relaciones inexistentes. Este último problema se debe principalmente a indetecciones de formas principales. Por ejemplo, si la clase *A* está relacionada con la *B* y esta a su vez con la *C* y se detectan *A* y *C* pero no *B*, es muy probable que aparezca relacionadas *A* con *C*. Se podrían paliar estos defectos con aumentos en los tamaños de las imágenes que permitiesen distinguir mejor los segmentos de menor longitud, de los textos sobre cardinalidades y otros elementos que dificultan su reconocimiento.

Por otra parte, se han encontrado defectos en el reconocimiento de formas representadas de forma no estándar. En este sentido estos defectos eran esperables y su repercusión se describe en la literatura. Aunque, en la acotación del trabajo de investigación no había compromiso de reconocer otras formas alternativas, se pone de manifiesto la sensibilidad de los métodos de reconocimiento frente a los cambios en las representaciones. Por ejemplo, no se detectan todos los rectángulos si estos están representados con las esquinas redondeadas. Tampoco se reconocen todas las relaciones si aparece un salto en su trazo para insertar texto o si en su trazado se emula un puente curvo para indicar un cruce entre relaciones. En cualquier caso, cualquier ampliación en este sentido requiere trabajar con todas las formas de representación que se pretendan reconocer, de no hacerse así las variantes de estas podrían reducir las tasas de acierto.

4.2 Discusión sobre el filtrado de imágenes de diagramas UML de procedencia web

En este bloque se ha propuesto una metodología para filtrar de forma automática las imágenes que realmente representan diagramas UML. El objetivo perseguido es el de no procesar aquellas imágenes que no representan diagramas con el fin de minimizar el tiempo de proceso. Por tanto, la clasificación de imágenes requiere tasas de acierto aceptables en un tiempo razonable. En el estado del arte se han descrito trabajos orientados a la clasificación de imágenes, sin embargo los tiempos de proceso los hacen inviables para el fin que se persigue.

En principio la validez de los modelos de aprendizaje, creados para la clasificación de las imágenes, dependerá de los resultados de evaluación, de su tiempo de respuesta, de los porcentajes de imagines positivas (diagramas) y negativas (no diagramas) esperados y del tiempo empleado en procesar imágenes negativas sin aplicar un filtrado previo.

Aunque en principio, por lo expuesto, habría que realizar un estudio dependiendo en cada caso de los parámetros indicados, los modelos construidos se juzgan aplicables de forma general. Esta afirmación, se apoya en sus buenos resultados de evaluación (tanto precisión como recall) y en su tiempo de respuesta. Dado que los modelos creados tienen a los sumo un par de decenas de reglas y cada una de ellas expresa de dos a cuatro comparaciones, el número medio de comparaciones por imagen es inferior a 30. Por tanto, la repercusión temporal de la aplicación de los modelos es despreciable.

Cabe resaltar que la evaluación de los modelos nos proporciona porcentajes sobre ambas categorías (imágenes positivas y negativas). El acierto global sobre un lote a procesar dependerá por tanto de la proporción de imágenes presentes de cada clase. Esta proporción será variable pues dependerá del motor de búsqueda elegido para recuperar imágenes de la Web y de los cambios que experimenten sus algoritmos.

En cuanto a valorar los beneficios de optimización temporal que aporta la aplicación del filtrado de imágenes es preciso localizar el momento en que se lleva a cabo. Tal como se ha probado es necesario obtener la información estructural de una imagen para poder clasificarla con garantías. Afortunadamente el tiempo medio invertido hasta ese punto es aproximadamente un 10% del tiempo medio total de proceso. Cuando se descarta una imagen por no representar un diagrama UML se evita realizar el reconocimiento de texto.

Por tanto, en estos casos el ahorro de tiempo de proceso puede alcanzar aproximadamente, dependiendo de la tipología de las imágenes, hasta el 90%.

4.3 Discusión sobre el reconocimiento del texto presente en imágenes UML

El tema de estudio de este bloque de investigación se ha centrado en el reconocimiento de la información textual presente en imágenes que representan diagramas UML.

El enfoque metodológico propuesto se ha dirigido principalmente a la mejora de la calidad de las imágenes web, mediante preprocesados, con miras a la aplicación de herramientas OCR. Tomando como punto de partida una investigación fruto de una colaboración se afinaron los algoritmos para alcanzar tiempos razonables de ejecución.

En definitiva se ha encontrado un modo de proceder extrapolable a otras herramientas OCR. Se ha puesto de manifiesto la sensibilidad de los reconocedores de texto al tipo de letra y a la resolución de la imagen. También se dan evaluaciones de distintos software OCR aplicados sobre ejemplos de los casos de estudio.

Sin embargo, a pesar de reconocer sólo el texto presente en el interior de los elementos gráficos, el tiempo necesario para la captura del texto es elevado en comparación con el empleado en la detección de formas. Además, los errores de reconocimiento de caracteres proporcionan un número significativo de palabras incorrectas. Este problema se agrava en las palabras más largas como los identificadores utilizados en el desarrollo de software combinando varios términos. Mención aparte merece el problema de la delimitación de las palabras reconocidas. Es habitual que se incluyan espacios en blanco entre los caracteres de una palabra.

Parece necesario un tratamiento posterior del texto mediante PLN y correctores ortográficos posiblemente apoyados en ontologías.

4.4 Extracto de las principales conclusiones

En este apartado se recogen las principales conclusiones a las que se ha llegado en esta investigación. Se pueden consultar con más detalle en los apartados de discusión y conclusiones correspondientes a cada uno de los tres bloques en los que se ha dividido la investigación.

- Se ha demostrado la posibilidad de capturar la información de modelos UML a partir de sus representaciones en formato imagen con tiempos de cómputo aceptables.
- Se abre la puerta a practicar reutilización del software automática a partir de un repositorio universal, la Web.
- Se han construido modelos por aprendizaje automático que permiten discernir de forma preliminar que imágenes representan modelos UML (de los casos de estudio).
- La eficiencia mostrada en algunas fases del desarrollo de la metodología son muy apropiadas. Sin embargo, el reconocimiento de caracteres consume aproximadamente el 90% del tiempo total.
- También hay margen de mejora en el reconocimiento de objetos de escaso tamaño. Las recomendaciones son que las líneas tengan al menos 20 píxeles para tener cierta certeza en su detección.
- Las representaciones no estándar de figuras UML pueden producir falsos positivos, al igual que podría ocurrir en el punto anterior.

Capítulo V: Líneas futuras de investigación

Existen diversas líneas por la que dar continuidad a esta investigación. Se relacionan a continuación las más señaladas:

- Incorporación de otros tipos de diagrama UML como casos de estudio y de otras representaciones de las formas trabajadas.
- Mejorar el reconocimiento de caracteres tanto en eficacia como eficiencia. La utilización de herramientas OCR licenciadas, PLN, ontologías y correctores ortográficos podrían ser opciones validas en este sentido.

  Por otra parte, los compromisos de eficiencia hacen que no se aplique el OCR en el exterior de las formas detectadas. Esto impide extraer información valiosa como la multiplicidad de los enlaces. Además, por el elevado coste computacional sólo se hace reconocimiento del texto orientado horizontalmente. Por estos y otros problemas resulta necesario conseguir una disminución del tiempo empleado por las herramientas OCR.
- Perfeccionamiento en la detección de conexiones entre elementos gráficos.

  Situaciones como la de la siguiente ilustración (rectángulo discontinuo sin vértices próximo a relación discontinua), en las que es difícil de obtener resultados idóneos, muestran la dificultad en la detección precisa de formas y sobre todo de relaciones. Es evidente que esta línea de investigación tendrá un largo recorrido.

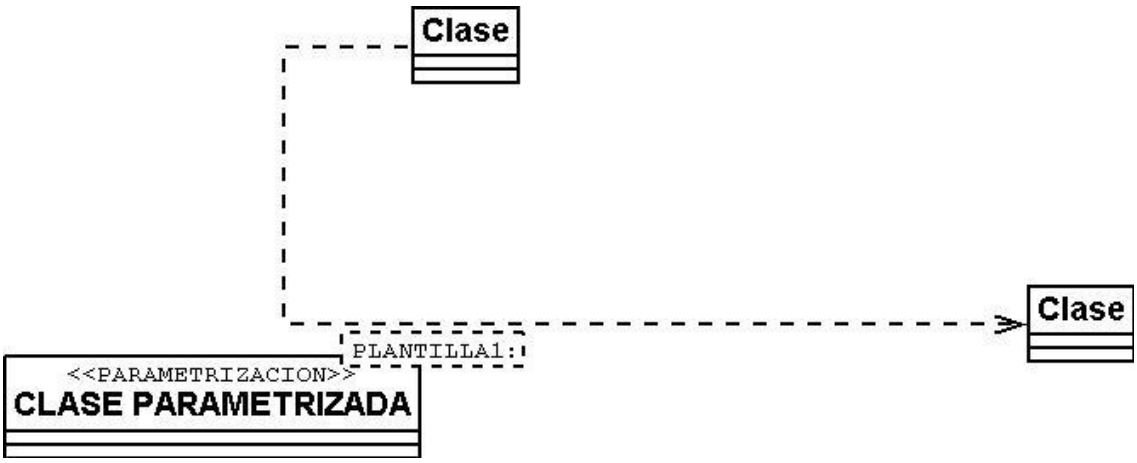


Ilustración V-1: Ejemplo de problemas con las clases parametrizadas.

- Detección y procesado de sub-diagramas. Es decir, diagramas contenidos en otros diagramas. Esta línea no parece a priori necesitar demasiados cambios metodológicos aunque resulta atractiva por que permitiría completar la extracción de información de los diagramas que presentan estas peculiaridades.

Referencias

[Antoniou y Harmelen, 2004] Antoniou, G. Y Harmelen, F. van. A Semantic Web Primer. London: The MIT Press, 2004.

[Bass et al., 2000] Bass, L.; Buhman, Ch.; Comella, S.; Long, F.; Robert, J.; Seacord, R.; Wallnau, K. Market Assessment of Component-based Software Engineering. Vol. I-May. 2000. Technical Note MU/SE-2001-TN-007.

[Bhat y Hammond, 2009] Bhat, A.; y Hammond, T. 2009. "Using Entropy to Distinguish Shape Versus Text in Hand-Drawn Diagrams". En International Joint Conference on Artificial Intelligence (IJCAI '09). Pasadena, California.

[Blagojevic et al., 2010] Blagojevic, R.; Plimmer, B.; Grundy, J.; y Wang, Y. 2010. "Building Digital Ink Recognizers using Data Mining: Distinguishing Between Text and Shapes in Hand Drawn Diagrams". Proceedings of IEA-AIE 2010.

[Booch et al., 1999] Booch, G., Rumbaugh, J., Jacobson, I., & Molina, J. J. G. (1999). El lenguaje unificado de modelado (Vol. 1). Addison-Wesley.

[Booch et al., 2005] Booch, G., Rumbaugh, J., Jacobson, I. 2005.Unified Modeling Language User Guide (The Second Edition) (The Addison-Wesley Object Technology Series). Addison-Wesley Professional. ISBN-13: 978-0-321-26797-9.

[Boehm, 1988] Boehm, B. W. (1988). A spiral model of software development and enhancement. Computer, 21(5), 61-72.

[Breiman, 1996] Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123-140.

[Cendrowska, 1987] Cendrowska J. (1987). PRISM: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 7 (4), 349-370.

[Chen, 1976] Chen, P.P. The Entity Relationship-Model: Toward a Unified View of Data. Transactions on Database Systems 1976, 1 (1). pp. 9-36

[Clark y Niblett, 1989] P. Clark and T. Niblett. "The CN2 induction algorithm. Machine Learning". 3: 261-283, 1989.

[Conklin, 2001] Conklin, J. (2001). Designing organizational memory: Preserving intellectual assets in a knowledge economy. Retrieved January 16, 2008, from http://cognexus.org

[Daconta et al., 2003] Daconta, Michael C.; Obrst, Leo J. y Smith, Kevin T. The Semantic Web. A guide to the future of XML, Web Services, and Knowledge Management. Indianapolis: Wiley, 2003.

[Damm et al., 2000a] Damm, C. H.; Hansen, K. M.; y Thomsen, M. 2000. "Tool support for cooperative object-oriented design: Gesture based modeling on an electronic whiteboard". En CHI 2000. CHI.

[Damm et al., 2000b] Damm, C. H.; Hansen, K. M.; Thomsen, M; y Tyrsted M. Junio 2000. "Creative Object-Oriented Modeling: Support for Intuition, Flexibility and Collaboration in CASE Tools", en ECOOP 2000 - Object-Oriented Programming: 14th European Conference, LNCS Volumen 1850, ed. E. Bertino. Págs. 27-43.

[Danielsson, 1990] Danielsson, P. E. (1990). "Generalized and Separable Sobel Operator" Machine Vision for three Dimensional Scene. . H. Freeman .Academic Press.

[Díaz, 2001] Díaz, I. Esquemas de Representación de Información basados en Relaciones. Aplicación a la Generación Automática de representaciones de Dominios. Director: Juan Llorens Morillo. Universidad Carlos III de Madrid, Departamento de Informática, 2001

[Dietterich, 1997] Dietterich, T. G. (1997). Machine learning research: four current direction. *Artificial Intelligence Magzine*, *4*, 97-136.

[Dietterich, 2000] Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. Machine learning, 40(2), 139-157.

[Dolk, 1984] Dolk, Daniel R.; Konsynski, Benn R. Knowledge Representation for Model Management Systems. By: IEEE Transactions on Software Engineering, Nov. 84, Vol. 10 Issue 6, p619-628, 10p, 8 diagrams, 2 color; (AN 14385941)

[Duffy, 2000] Duffy, J. (2000). Knowledge management: To be or not to be? Information Management Journal, 34(1), 64-67.

[Dutton, 1997] Dutton, E. S. (1997). Effects of knowledge reuse on the spacecraft development process (Doctoral dissertation, Massachusetts Institute of Technology).

[Escalera, 2001] Escalera Hueso, A. (2001). Visión por computador. Fundamentos y métodos. Prentice Hall.

[Frank y Witten, 1998] Eibe Frank and Ian H. Witten (1998). Generating Achúrate Rule Sets Without Global Optimization. In Shavlik, J., ed., *Machine Learning: Proceedings of the Fifteenth International Conference*, Morgan Kaufmann Publishers, San Francisco, CA.

[Fensel et al., 2002] Fensel; Bussler; Ding; et al. Semantic Web Application Areas. Proceedings of the 7th International Workshop on Applications of Natural Language to Information Systems. Stockholm, Sweden. 2002

[Fischler y Bolles, 1981] Fischler, M.A., Bolles, R.C. "Random sample consensus - a paradigm for model fitting with applications to image analysis and automated cartography", Comm. of the ACM 24 (1981) 381-395

[Flasiński, 1995] Flasiński, M. (1995). The Programmed Grammars and Automata as Tools for a Construction of Analytic Expert Systems, Archives of Control Sciences 40 (1995), 5–35

[Flasiński, 2002] Flasiński, M. (2002). Automata-based multi-agent model as a tool for constructing real-time intelligent control systems. In From Theory to Practice in Multi-Agent Systems (pp. 103-110). Springer Berlin Heidelberg.

[Frakes et al., 1998] Frakes, W., Prieto, R., & Fox, C. (1998). DARE: Domain analysis and reuse environment. *Annals of Software Engineering*, *5*(1), 125-141.

[Freund y Schapire, 1995] Freund, Y., & Schapire, R. E. (1995, January). A desicion-theoretic generalization of on-line learning and an application to boosting. In Computational learning theory (pp. 23-37). Springer Berlin Heidelberg.

[Freund y Schapire, 1996] Freund, Y., & Schapire, R. E. (1996, July). Experiments with a new boosting algorithm. In *ICML* (Vol. 96, pp. 148-156).

[García et al., 2008] Héctor García, Jorge Morato, Eugenio Santos, Gonzalo Génova (2008). Enabling Knowledge Reuse through Total Traceability in the context of Software Development. 10th International Conference on Software Reuse (ICSR), First Workshop on Knowledge Reuse (KREUSE 2008), 25-29 may 2008 Beijing China. Proceedings p. 17-21.

[Gevers y Groen, 1991] Gevers, T., & Groen, F. C. A. (1991). Segmentation of color images. In*Proceedings of 7th Scandinavian Conference on Image Analysis* (Vol. 1991)

[Godin et al., 1995] R Godin, G Mineau, R Missaoui, H Mili. Méthodes de classification conceptuelle basées sur les treillis de Galois et applications. Revue d'Intelligence Artificielle, 1995

[Gómez-Pérez et al., 2004] Gómez-Pérez, Asunción; Fernando-López, Mariano; Corcho, Oscar. Ontological Engineering. London: Springer, 2004.

[González, 1999] González, J. (1999). Visión por computador. ITP Paraninfo.

[Hammond y Davis, 2006] Hammond, T; y Davis, R. 2006. "Tahuti: A Geometrical Sketch Recognition System for UML Class Diagrams". Proceeding of ACM SIGGRAPH 2006 Courses. Boston, Massachusetts. Artículo Nº 25.

[Hjaltason y Samúelsson, 2015] Hjaltason, J., & Samúelsson, I. (2015). Automatic classification of UML Class diagrams through image feature extraction and machine learning.

[Hoetzlein, 2007] Hoetzlein, Rama (2007) Quanta. The Organization of Human Knowedge: Systems for Interdisciplinary Research. Master's Thesis, University of California Santa Barbara, June 2007

[Hong et al., 1986] J. Hong, I. Mozetic, and R. S. Michalski. AQ15: Incremental learning of attribute-based descriptions from examples, themethod and user's guide. *In Report ISG 85-5, UIUCDCS-F-86-949,* Department of Computer Science, University of Illinois at Urbana-Champaign, 1986.

[Hough, 1962] Hough, P. (1962). Methods and means for recognizing complex patterns. US patent 3069654.

[Hse et al., 1999] Hse, H.; Shilman, M.; Newton, A. R.; y Landay, J. 1999. “Sketch-based user interfaces for collaborative object-oriented modeling”. Berkley CS260 Class Project.

[Huertas, 1986] Huertas, A. a. (1986). “Detection od intensity changes using Laplacian-Gaussian Masks”. . IEEE Trans. Patern Analysis and Machine Intelligence.

[Ibison et al., 1993] Ibison, P.; Jacquot, M.; Kam, F.; Neville, A. G.; Simpson, R. W.; Tonnelier, C. A. G.; Venczel, T.; y Johnson, A. P. 1993. “Chemical literature data extraction: The clide project” Journal of Chemical Information and Computer Sciences 33 (Volumen 3). Págs. 338–344

[Jacobson et al., 1999] Jacobson, I., Booch, G., Rumbaugh, J., Rumbaugh, J., & Booch, G. (1999).*The unified software development process* (Vol. 1). Reading: Addison-wesley.

[Jarzabek, 1993] Jarzabek, S.: “Domain Model-driven Software Reengineering and Maintenance”. The Journal of Systems and Software, vol 20 (1), pp 37-51, 1993.

[Karasneh y Chaudron, 2013a] Karasneh, B., & Chaudron, M. R. (2013, March). Extracting UML models from images. In *Computer Science and Information Technology (CSIT), 2013 5th International Conference on* (pp. 169-178). IEEE.

[Karasneh y Chaudron, 2013b] Karasneh, B., & Chaudron, M. R. (2013, September). Img2uml: A system for extracting uml models from images. In *Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on* (pp. 134-137). IEEE.

[Karlsson, 1995] Karlsson, E. A. (1995). Software reuse: a holistic approach. John Wiley & Sons, Inc..

[Landay y Myers, 1995] Landay, J.A.; y Myers, B.A. 1995. “Interactive sketching for the early stages of user interface design”. En CHI. Págs. 43-50

[Lank et al., 2000] Lank, E.; Thorley J.; y Chen, S. Noviembre 2000. “An Interactive System for Recognizing Hand Drawn UML Diagrams”. Center for Advanced Studies Conference 2000 (CASCON 2000) Mississuaga, Ontario. Págs 1-15.

[Lank et al., 2001] Lank, E.; Thorley J.; Chen, S.; y Blostein D. Septiembre 2001. “On-line Recognition of UML Diagrams”. en Six International Conference on Document Analisys and Recognition. Preecedings. Seattle, Washington. Págs 356-360.

[Lank, 2001] Lank, E. Abril 2001. “Retargetable On-line Recognition of Diagram Notations”. Tesis doctoral. Department of Computing and Information Science, Queen’s University, Kingston, Canada.

[Lin et al., 2001] Lin, J.; Newman, M.W.; Hong, J. I.; y Landay, J.A. 2001. “Denim: An informal tool for early stage website design”. En Video poster en Extended Abstacts of Human Factors in Computing Systems: CHI 2001. Págs 205-206

[Llorens et al., 2004] Llorens, Juan; Morato, Jorge; Génova, Gonzalo (2004). RSHP: an information representation model based on relationships. Soft-Computing in Software Engineering: Theory and Applications. LNCS series (Studies in Fuzziness and Soft Computing, v. XIII). Agosto 2004. p 221-253. Ed. Damiani, Ernesto; Jain, Lakhmi C.; Madravio, Mauro. Springer Verlag, New Cork. ISBN: 3-540-22030-5.

[Llorens y Velasco, 1995] Llorens, J., Amescua, A. & Velasco, M.: “Software Reusability: The Need to Modify the Software Analysis Methodologies and the Repository Structure”. Gupta World. 1995.

[Llorens, 1996] Llorens, J.: “Definición de una Metodología y una Estructura de Repositorio orientadas a la Reutilización: el Tesauro de Software”. Teis Doctoral. Universidad Carlos III de Madrid. 1996.

[Major y Mangano, 1995] Major, J. A., & Mangano, J. J. (1995). Selecting among rules induced from a hurricane database. *Journal of Intelligent Information Systems*, *4*(1), 39-52.

[McDaniel y Balmuth, 1992] McDaniel, J. y Balmuth, J. 1992. “Kekulé: Ocr-optical chemical (structure) recognition”. Journal of Chemical Information and Computer Sciences 32 (Volumen 4). Págs. 373-378

[Menzies y Justin, 2003] Tim Menzies and Justin S. Di Stefano. More Success and Failure Factors in Software Reuse. IEEE Transactions on Software Engineering, Vol. 29, No. 5, May 2003

[Mitov, 2008] Mitov (2008). Mitov Software. Obtenido de www.mitov.com

[Moreno et al., 2006] Moreno, V., Ledezma, A., & Sanchis, A. (2006, February). A Static Images Based-System for Traffic Signs Detection. In *Artificial Intelligence and Applications* (pp. 445-450).

[Morisio et al., 2002] Maurizio Morisio, Michel Ezran, Colin Tully. Success and Failure Factors in Software Reuse. IEEE Transactions on Software Engineering, v.29 n.5, p.474-477, 2002

[Neighbors, 1981] Neighbors, J.: "Software Construction using Components". Ph. D. Thesis. Department of Information and Computer Science. University of California, Irvine, 1981.

[Neighbors, 1984] Neighbors, J.: "The Draco Approach to Constructing Software from Reusable Components". IEEE Transactions on Software Engineering. Vol SE-10 (5), 1984.

[Ning, 1993] Ning, J. Q., Engberts, A. & Kozaczynski, W. : "Recovering Reusable Components from Legacy Systems by Program Segmentation". Working Conference on Reverse Engineering, pp 64-72. IEEE Computer Society Press, 1993.

[Nonaka y Takeuchi, 1995] Nonaka, I., & Takeuchi, H. (1995). The knowledge-creating company: How Japanese companies create the dynamics of innovation. Oxford university press.

[Patel et al., 2007] Patel, R., Plimmer, B., Grundy, J., & Ihaka, R. (2007, August). Ink features for diagram recognition. In Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling (pp. 131-138). ACM.

[Plimmer y Freeman, 2007] Plimmer, B.; y Freeman, I. 2007. "A Toolkit Approach to Sketched Diagram Recognition". HCI, Lancaser, Reino Unido. Págs. 205-2013.

[Prieto-Díaz, 1987] Prieto-Díaz, R.: "Domain Analysis for Reusability". Proceedings of COMPSAC'87, pp 23-29, Tokyo, 1987.

[Prieto-Díaz, 1990] Prieto-Díaz, R.: "Domain Analysis: An Introduction". ACM Sigsoft. Software Engineering Notes. Vol 15(2). 1990.

[Prieto-Díaz, 1991] Prieto-Díaz, R.: "Implementing Faceted Classification for Software Reuse". Communications of the ACM. Vol 34(5). 1991.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of Decision Trees (ID3 algorithm). *Machine Learning*, *1*(1), 81-106.

[Quinlan, 1993] Quinlan, J. R. (1993). C4.5: Programs for Machine Learnirig. *Morgan Kaufmann, CA*.

[Radon, 1917] Radon, J. "Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten", Berichte und Verhandlungen der Sächsischen Akademie der Wissenschaften, Math./Nat. Klasse 69 (1917) 262-277

[Ramesh y Jarke, 2001] Balasubramaniam Ramesh y Matthias Jarke (2001). Towards Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering, vol. 27, issue 1, pp. 58-93. IEEE. 2001.

[Robert, 1965] Robert, L. (1965). "Machine perception of Three Dimensional Solids". Cambridge: In Optical and electroptical Information Processing, J.T.Trippet et al.The MIT Press.

[Rumbaugh et al., 1998] Rumbaugh, J.; Jacobson, I. y Booch, G. The unified modeling language reference manual. Massachusetts: Addison-Wesley, 1998.

[Sadawi et al., 2011] Sadawi, N. M.; Sexton, A. P.; y Sorge, V. 2011. "Performance of MolRec at TREC 2011 – overview and analysis of results". En Proceeding of 20th TREC. NIST

[Sadawi et al., 2012] Sadawi, N.M.; Sexton, A.P.; y Sorge, V. Enero 2012. "Chemical Structure Recognition: A Rule Based Approach". Document Recognition and Retrieval XIX. Volumen 8297. Burlingame, California.

[Schapire, 1990] Schapire, R. E. (1990). The strength of weak learnability. *Machine learning*,*5*(2), 197-227.

[Valko y Johnson, 2009] Valko, A. T., & Johnson, A. P. (2009). CLiDE Pro: the latest generation of CLiDE, a tool for optical chemical structure recognition. Journal of chemical information and modeling, 49(4), 780-787.

[Van den Bosch, 2014] Van den Bosch, P. Enero 2014. "Automation of use case diagram recognition from images". Tesis de licenciatura. Leiden Institute of Advanced Computer Science (LIACS). Leiden, Holanda. http://www.liacs.nl/assets/Bachelorscripties/2013-2014PietervandenBosch.pdf

[Velasco, 1998] Velasco De Diego, M. Generación Automática de Representaciones de Dominios. Director: Vicente Martínez Orga. Universidad Politécnica de Madrid, Facultad de Informática. 1998.

[Vernon, 1991] Vernon, D. (1991). "Machine Vision. Automated Visual Inspection and Robot Vision. Prentice Hall.

[Voss et al., 2004] Voss, K., Suesse, H., & Ortmann, W. (2004). Radon, Hough, Acumulación y el método SDR. CC/CIMAT Dep. Mathematic, Comunicación Técnica No I-04-05.

[Voss y Suesse, 2001] Voss, K., Suesse, H. "Apareamiento de puntos y optimización lineal", Comunicaciones del CIMAT/Guanajuato/ México, Comunicación Técnica 1-01-06/05-06-2001.

[Weiss y Indurkhya, 1998] Weiss, S. M., & Indurkhya, N. (1998). Predictive data mining: a practical guide. Morgan Kaufmann.

[Weiser y Morrison, 1998] Weiser, M., & Morrison, J. (1998). Project memory: Information management for project teams. Journal of Management Information Systems, 14(4), 149-166.

[Witten y Frank, 2000] Witten, I. H., & Frank, E. (2000). Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann.

[Witten y Frank, 2005] Witten, I. H., & Frank, E. (2005). Data Mining. Practical Machine Learning Tools and Techniques, 2th Ed. *Morgan Kaufmann Publishers*. Ed. Morgan Kaufmann Publishers.

[Wolpert, 1992] Wolpert, D. H. (1992). Stacked generalization. *Neural networks*, *5*(2), 241-259.

Anexo A. Estudio de software OCR aplicados a imágenes web

INFORME FINAL PROYECTO UML

Madrid Febrero 2008

Introducción

El presente informe recoge los resultados finales del proyecto "Reconocimiento de caracteres en Gráficos UML", cuyo objetivo consiste en el **estudio de métodos para mejorar el reconocimiento de caracteres en los diagramas UML.**

El estudio realizado se encuentra dividido en 3 partes. La primera parte presenta un análisis de los procesamientos de imágenes para la mejora de la tasa de reconocimiento de caracteres, utilizando para ello como OCR, la herramienta MODI de Microsoft. La segunda parte, encierra un estudio comparativo de los diferentes programas disponibles en el mercado para el reconocimiento de caracteres. En la tercera parte se extraen las conclusiones de los trabajos realizados con recomendaciones concretas encaminadas a la implementación eficiente de un OCR dentro de los diagramas UML.

Los estudios realizados se han basado en un conjunto de 30 imágenes con características distintas, que han sido elegidas como representativas de distintos tipos de acabado y tipo de letra y cuyas resoluciones corresponden a las más frecuentes en el estándar Web (entre 72 dpi y 96 dpi). Este conjunto de imágenes se recogen en el ANEXO I.

Preprocesado de imágenes para la mejora de resultados de un OCR.

El objetivo de esta primera parte es encontrar los algoritmos de procesamiento de imágenes que permitan mejorar la tasa de reconocimiento de un OCR, utilizando para ello el estándar MODI aplicado a la base de datos anteriormente descrita. Un primer estudio realizado en el mes de noviembre, arrojó resultados bastante satisfactorios sobre la mejora en la tasa de aciertos del reconocedor, al aplicar un preprocesado a la imagen basado en el remuestreo. En dicha ocasión se analizaron las tasas de reconocimiento obtenidas al aplicar tres técnicas de remuestreo (bilineal, bicúbico y por similitud) a diferentes escalas (2, 4 y 6 veces el tamaño original de la imagen). Las conclusiones encontradas en dicho estudio resaltan la notoria mejora que se obtiene en el reconocimiento al aplicar diferentes técnicas de remuestreo especialmente en aquellas imágenes donde las características de la imagen original (cantidad de ppi y acabado entre otras) impiden que el software ejecute un reconocimiento de caracteres (Img5, Img16, Img17, Img19, Img23, Img27, Img29).

Este primer análisis realizado muestra claramente que la primera etapa que debe cumplir el algoritmo, tiene que ver con asegurar que en las imágenes se pueda ejecutar un reconocimiento de caracteres independientemente de la calidad del mismo, el cual se procederá a mejorar en las etapas subsecuentes.

Microsoft presenta una herramienta para desarrolladores la cual fue utilizada durante este estudio, analizándose los diferentes resultados al modificar los parámetros disponibles para el reconocimiento de caracteres. En esta herramienta el texto es reconocido mediante un software de reconocimiento de formas y estructuras que compara los caracteres del texto digitalizado con el diccionario integrado de secuencias y formas de caracteres. El diccionario proporciona todas las letras en mayúsculas y minúsculas, signos de puntuación y acentos utilizados en el idioma seleccionado. De forma predeterminada, MODI utiliza el diccionario del idioma que usan las otras aplicaciones de Microsoft Office, el cual puede ser modificado por el usuario. Los otros dos parámetros a controlar para el reconocimiento de caracteres son:

- Auto-girar: Esta opción permite girar el documento si ha sido digitalizado de lado o al revés, colocando la página en la posición correcta

- Auto-enderezar: Si el papel se ha digitalizado un poco desajustado, la selección de esta opción vuelve a colocar la página en la posición correcta.

Otra información adicional sobre cómo mejorar el reconocimiento, se recogió a través de la información disponible en la librería; estos requerimientos adicionales se resumen a continuación:

Para obtener la máxima precisión de OCR, utilizar imágenes con las siguientes características:

- Monocromáticas a 300 dpi.

- Imágenes en escala de grises a 200 dpi

Cuanto más grande sea el número de puntos por pulgada, mayor será el archivo resultante. Además, los archivos digitalizados con una resolución demasiado alta, algunas veces dan problemas en el proceso OCR. Para obtener los mejores resultados conviene utilizar una resolución de digitalización entre 300 y 400 puntos por pulgada.

Establecer un idioma para el OCR

A.1.1 Pruebas realizadas

Las pruebas realizadas con la herramienta MODI se dividen en 2 etapas, las cuales son:

Etapa I: Pruebas de procesamiento de imágenes.

Etapa II: Procesamiento ligado a características de la imagen

En los siguientes párrafos se describirán cada una de las etapas anteriores

5.1.1.1 Etapa I: Pruebas de procesamiento de imágenes

Esta primera etapa encierra un conjunto de pruebas realizadas a las imágenes de la base de datos disponible, con el fin de analizar el comportamiento del OCR ante las diferentes configuraciones de sus parámetros e igualmente ante los diferentes procesamientos que se pueden hacer a las imágenes.

Las principales pruebas realizadas se resumen en la Tabla A-1; los resultados detallados de las mismas se pueden ver en el 171.

La Ilustración A-1 (ANEXO VII) encierra los resultados obtenidos en cada una de las pruebas realizadas. Es importante aclarar que todos los resultados expuestos en el documento hacen referencia a porcentaje de letras reconocidas.

Prueba	Parámetros de OCR			Canal Analizado	Remuestreo	Preprocesado Adicional
	Idioma	Autoenderezar	Autogirar			
1	Por Defecto	True	True	RGB	NA	NA
2	Inglés	False	False	RGB	NA	NA
3	Inglés	False	False	R	NA	NA
4	Inglés	False	False	G	NA	NA
5	Inglés	False	False	B	NA	NA
6	Inglés	False	False	Escala Grises	NA	NA
7	Inglés	False	False	**S** (HSV)	NA	NA
8	Inglés	False	False	**V** (HSV)	NA	NA
9	Inglés	False	False	RGB	2X Bicubic	NA
10	Inglés	False	False	RGB	4X Bicubic	NA
11	Inglés	False	False	RGB	6X Bicubic	NA
12	Inglés	False	False	RGB	300 ppi	NA
13	Inglés	False	False	R	300 ppi	NA
14	Inglés	False	False	R	300 ppi	Erosión
15	Inglés	False	False	R	300 ppi	Erosión
16	Inglés	False	False	R	300 ppi	Dilatación
17	Inglés	False	False	R	300 ppi	Dilatación

Tabla A-1: Pruebas realizadas Etapa 1:"Pruebas de procesamiento de imágenes"

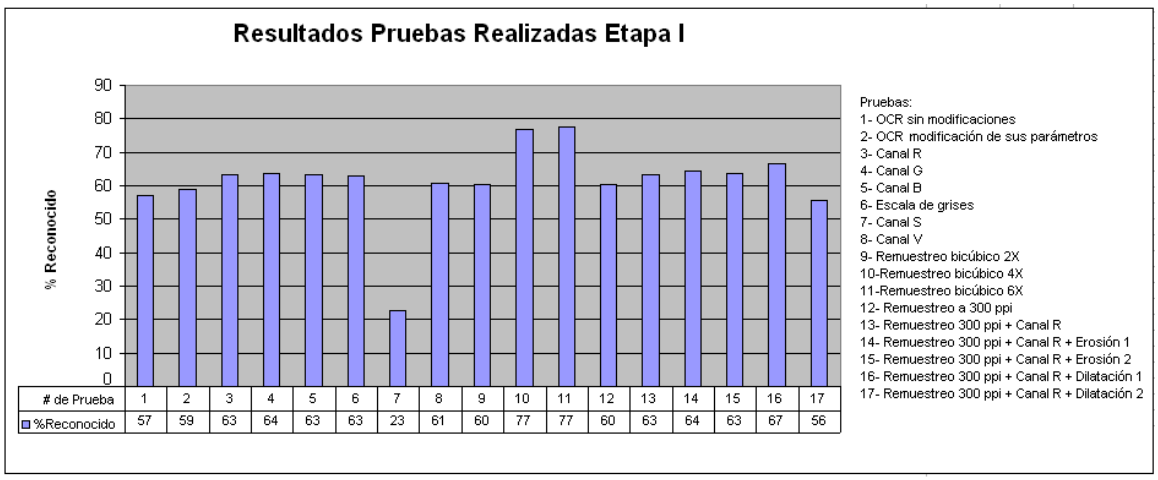


Ilustración A-1: Resultados pruebas realizadas Etapa I:"Pruebas de procesamiento de imágenes"

Las pruebas realizadas han permitido ver la posibilidad de mejorar la tasa de reconocimiento de la herramienta MODI al modificar tanto sus parámetros (idioma, auto-enderezar y auto-girar) como las características de la imagen. Lo expuesto anteriormente se puede ver al analizar las imágenes 18, 12 y 6 donde al modificar los parámetros del OCR se pasa de reconocimientos de alrededor de un 82% a un 100% e igualmente en las imágenes 9, 10, 25 y 30 donde se consiguen mejoras bastante significativas en la tasa de reconocimiento al modificar las características de la imagen.

Las siguientes figuras ilustran los casos mencionados anteriormente.

Imagen	Resultados OCR parámetros por defecto	Resultados OCR modificando parámetros (Ingles,False,False)
Name Company Address Telephone Email	Name C ompany Aciciress Telephone Email **94%**	Name Company Address Telephone Email **100%**

Ilustración A-2: Resultados del reconocimiento de caracteres en la imagen 6.

La Ilustración A-2 permite ver las mejoras obtenidas en la tasa de reconocimiento al modificar los parámetros del OCR; esta mejora se obtiene seleccionando como idioma el

inglés y ajustando los otros dos parámetros (auto-enderezar y auto-girar) en FALSE; es importante destacar que al dejar estos parámetros en FALSE se asegura que la imagen no sea girada antes de aplicar el OCR; esto cobra importancia al realizarse el remuestreo, ya que dependiendo del nuevo tamaño de la imagen, el software podría rotar la imagen haciendo que la información reconocida sea errónea.

En la Ilustración A-3 se presenta un ejemplo de este caso donde al aplicar un remuestreo de 8x a la imagen; se pueden ver los resultados obtenidos del OCR ante las dos posibles opciones de los parámetros (True/False) y el esperado mal desempeño en el reconocimiento (debido a lo comentado anteriormente) cuando la opción auto-girar esta en True.

Imagen	Resultados Remuestreo Bicúbico 8X (Ingles,False,False)	Resultados Remuestreo Bicúbico 8X (Ingles,True,True)
+ ABall(Point p, int r, Point v, Color c, Container container) + void : update(Observable o, Object g) + void : bounce() + void : setLocation(Point location) + Point : getLocation() + void : setRadius(int radius) + int : getRadius() + void : setVelocity(Point velocity) + Point : getVelocity() + void : setColor(Color color) + Color : getColor() + void : paint(Graphics g) + *void : updateState()*	ABall(Point p. mt r. Point v. Color c. Container container) void : updateçObservable o. Object g) + ;'oid : void : setLocationçPoint location) Point : etLocation() void : setRadius(int radius) mt : getRadiusO void : setVelocitvçPoint velocity) Point : getVelocity() void : setColor(Color color) + Color : getColor() void, : paintçGraphics g) vosd uzaaleSlale'.' a, a	1 ,4 . d £:4 Q' a O r 0 0 ..4 QA O

Ilustración A-3: Resultados del reconocimiento de caracteres en la imagen 26.

Imagen	Resultados OCR parámetros por defecto	Resultados OCR modificando parámetros	Resultados Escala de grises	Resultados Remuestreo Bicúbico 2X	Resultados Remuestreo Bicúbico 6X
accountName : string nickname : string fullName : string emailAddress : string organization : string phoneNumber : string postalAddress : string fax : string image : string	accountName stnng nickname : sn g fuliName : stnng emailMdress string orgarizaton stnng phoneNumber : siñng ɔost3UJddress : sing x :string image string **83%**	account Name string nickname : shin 9 fuliName : string emailMdress string org ariiz air on string phoneNumber : siring postalPddress : siring x string image string **88%**	account Name : string nickname : shin 9 fillName : string emailMdress string organizaton :string phone Number : string postaiddress : string x string image string **90%**	account Name: string nickname : sbing fuilName : string emailAddress : string organizaon : string phoneNumber : sthng postaldress : sting fax : stnng image :stnng **90%**	account Name: Etnng nickname : string fuilName : string emailAddress : string organization :tring phone Number : string potalAidress : string fax :string image :string **96%**

Ilustración A-4: Resultados del reconocimiento de caracteres en la imagen 8.

Por otro lado, en las ilustraciones Ilustración A-4 y Ilustración A-5 se puede observar la mejora en la tasa de reconocimiento al aplicar diferentes preprocesados a la imagen; de la Ilustración A-4 se destaca no solo la mejora obtenida al modificar los parámetros del OCR; sino a su vez la mejora en la tasa de reconocimiento al convertir la imagen original (RGB) en escala de grises, en otros canales de color (HSV) o tomando un solo de la imagen (RGB ó HSV).

Otro aspecto importante arrojado con las diferente pruebas desarrolladas en esta etapa es la importancia del remuestreo para asegurar que el MODI reconozca información en la imagen; en este punto se puede analizar el caso de las imágenes 5, 16, 17, 19, 23 y 28, donde sin modificar las características de la imagen, no es posible reconocer información en ellas; pero una vez realizado un remuestreo es posible ejecutar el OCR.

En la Ilustración A-5 se puede ver como gracias al remuestreo es posible no solo lograr un reconocimiento de caracteres en la imagen; sino adicional a esto, una mejora en el porcentaje de reconocimiento al seleccionar una adecuada tasa de remuestreo; por otro lado es importante añadir que dicho remuestreo deberá ser controlado para evitar el efecto contrario al esperado en la tasa de reconocimiento; esto se puede ver en la Ilustración A-5 donde al aplicar un remuestreo bicúbico de 5X la calidad de la información reconocida desmejora notoriamente.

Imagen	Resultados OCR parámetros por defecto	Resultados OCR modificando parámetros	Resultados Remuestreo Bicúbico 2X	Resultados Remuestreo Bicúbico 3X	Resultados Remuestreo Bicúbico 5X
+handleClick(mode : int)	No se puede ejecutar	No se puede ejecutar	+handleCleck(rnode: int) 90%	+handleClick(rro'de: int) 95%	+hafldllick(a)cxe int) 71%

Ilustración A-5: Resultados del reconocimiento de caracteres en la imagen 19.

Con base en lo expuesto hasta el momento se puede concluir que la primera parte del algoritmo de reconocimiento de caracteres debe estar relacionada con asegurar que se efectúe un reconocimiento en la imagen, independientemente de los resultados del mismo; lo cual se puede obtener de forma efectiva a través de un resmuestreo progresivo 2X, 3X … 8X, con los diferentes tipos de interpolación (bicubica, bilineal y por similitud) hasta que se logre un reconocimiento de caracteres en la imagen. Se plantea la opción de un remuestreo progresivo debido a la complejidad del problema, al no tener información sobre el tipo de imagen que debe abordar el reconocedor.

Adicional a esto, las pruebas dejan claro la mejora que se logra en el reconocimiento de caracteres al modificar los parámetros del OCR (Ingles,False,False) y al utilizar un solo canal de la imagen (ver Ilustración A-1). Los resultados obtenidos son similares al usar cualquiera de los métodos de pasar a nivel de gris (ver Ilustración A-6), sin embargo teniendo en cuenta que el número de pruebas es limitado y que los OCR recomiendan trabajar con las imágenes en escalas de grises usando la luminosidad de la imagen; se toma dicha luminosidad como mejor opción para realizar las pruebas.

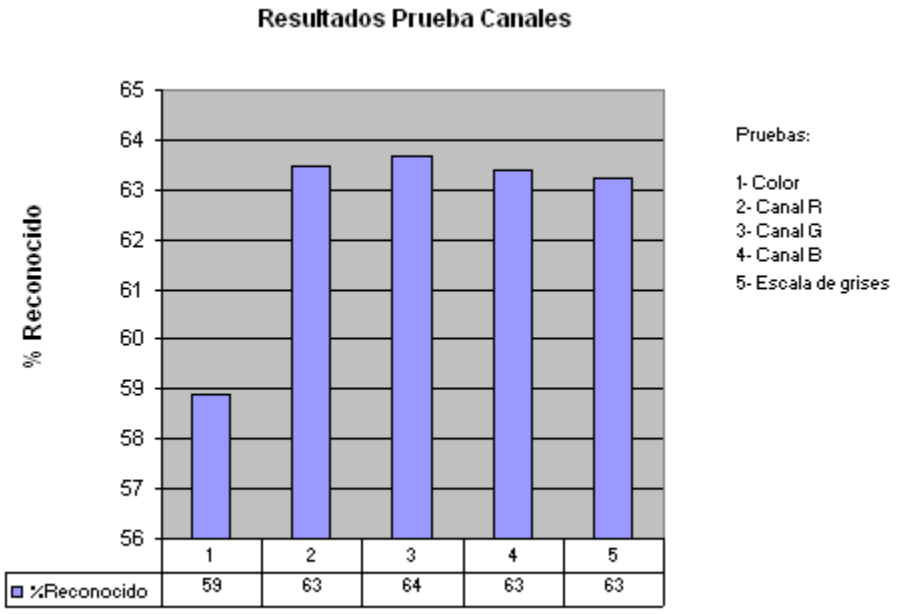


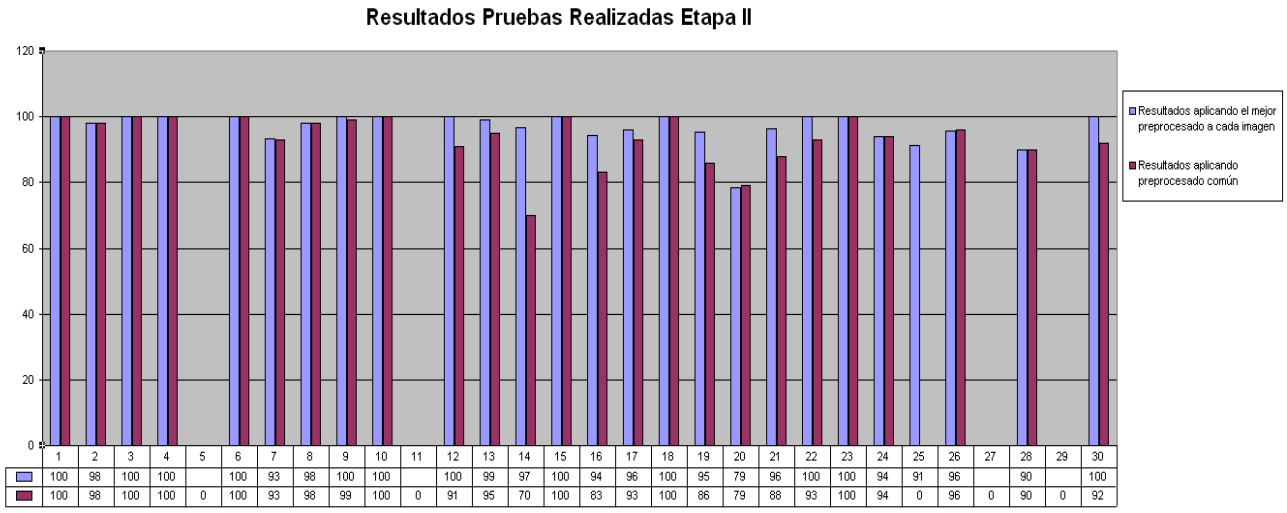
Ilustración A-6: Resultados del reconocimiento de caracteres con diferentes representaciones de la imagen (color, canal R, canal G canal B y luminosidad para escala de grises)

5.1.1.2 Etapa II: Procesamiento ligado a características de la imagen.

Una vez concluidas las pruebas de la etapa I y analizados los datos obtenidos; la etapa II consistió en analizar, a través de la combinación de las diferentes pruebas que arrojaron las mejores tasas de reconocimiento, bajo qué circunstancias o bajo qué tipo de preprocesado se logra un reconocimiento lo más cercano posible al 100% para cada una de las imágenes; esto con el fin de tratar de establecer un preprocesado común que coincida con ciertas características extraídas de la imagen y que proporcione el mejor desempeño en el reconocimiento. Los resultados de estas pruebas se encuentran en el

ANEXO III. Las pruebas fueron realizadas con las imágenes en escala de grises y con los parámetros del reconocedor modificados de acuerdo a los resultados de la etapa anterior.

Con estas pruebas es posible analizar 2 tipos de información; la primera información consiste en determinar si existe un procesamiento común para las imágenes basado en el estilo de la fuente presente en las mismas, mientras que la segunda consiste en ver si es posible encontrar un tamaño ideal de letra que permita realizar un adecuado reconocimiento lo cual ayudaría a definir una proporción de remuestreo basada en esta característica.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
	100	98	100	100		100	93	98	100	100		100	99	97	100	94	96	100	95	79	96	100	100	94	91	96		90		100
	100	98	100	100	0	100	93	98	99	100	0	91	95	70	100	83	93	100	86	79	88	93	100	94	0	96	0	90	0	92

Ilustración A-7: Resultados al aplicar el procesamiento que arrojo los mejores resultados de reconocimiento comparado con los resultados al aplicar un procesamiento común basado en el FaceStyle

La Ilustración A-7 (ANEXO VIII) describe los resultados de las dos pruebas más importantes en esta etapa; en color celeste se encuentran los resultados obtenidos al aplicar los diferentes tipos de procesamiento a las imágenes que permitieron las mejores tasas de reconocimiento para cada una de las mismas; en color púrpura se encuentran los resultados obtenidos al aplicar un procesamiento común (ver Tabla A-3) basado en la información extraída en las imágenes, las conclusiones de estas pruebas se explican a continuación.

Si se analiza las imágenes 6 y 12 cuyo tamaño de fuente es 11 de acuerdo a la tabla del ANEXO III; es posible ver que en éstas no es necesario realizar un remuestreo para

obtener unos resultados del 100%; con lo cual se puede rescatar la importancia del tamaño de la fuente a la hora de ejecutar el OCR. Esta información comparada igualmente con las imágenes cuyos resultados alcanzaron un 100% permite concluir que en tamaños de letra inferiores a 10 píxeles, un incremento de su tamaño a 16 o mayor mejora notoriamente el resultado del reconocimiento.

Por otro lado dicha tabla permite ver que dependiendo del estilo de la fuente y el tamaño en píxeles de la misma, es posible encontrar un procesamiento similar que permite alcanzar tasas de reconocimiento altas en todas las imágenes.

Para analizar lo expuesto anteriormente, en la Tabla A-2 se extrae del ANEXO III la información correspondiente a las imágenes con estilo de fuente Negrilla y CursivaNegrilla.

Imágenes	% Reconocido	Preprocesado		FaceStyle	Tamaño fuente Inicial
		Canal	Remuestreo		
Img7	93	Gris	Bicubic 3X	Negrilla	8
Img8	98	Gris	Bilinear2X	Negrilla	8
Img15	100%	Gris	Bicubic2X	CursivaNegrilla	8
Img20	79	Gris	Bicubic 3X-2X	Negrilla	10
Img25	91	Gris	Bicubic3X+Dilate	CursivaNegrilla	10
Img28	90	Gris	Bicubic 3X-2X	CursivaNegrilla	

Tabla A-2: Información extraída al aplicar un preprocesado a la imagen que permitiera las mejores tasas de reconocimiento.

En la tabla anterior se encuentra la información extraída de las imágenes al aplicar un procesamiento que arroja las mejores tasas de reconocimiento para cada una de ellas; entre la información que se puede extraer al pasar el OCR está el FaceStyle y el tamaño de fuente; como se mencionó anteriormente esta información es de gran importancia ya que permite plantear la opción del remuestreo y su proporción basados en dicha

información. Para el caso de la Tabla A-2, se puede ver como para un FaceStyle negrilla y un tamaño de letra inferior a 11 píxeles con un remuestreo de 3 veces el tamaño original de la imagen se consigue muy buenos resultados en la tasa de reconocimiento. Por otro lado se puede igualmente concluir que una interpolación bicúbica es la que mejor resultado da cuando se trata de un FaceStyle negrilla.

Imagen	Resultados OCR parámetros por defecto	Resultados con el mejor procesamiento	Resultados con procesamiento común
BodyCell	BodyCeli **88%**	BodyCell **100%**	BodyCell **100%**
AComponent	No se puede ejecutar	AComponen.t **100%**	AComprnen.t **90%**

Ilustración A-8: Resultados del reconocimiento de caracteres en las imágenes 18 y 28 respectivamente.

La Ilustración A-8 muestra los resultados obtenidos con el mejor procesamiento para las imágenes 18 y 28 e igualmente el resultado al aplicar el procesamiento común encontrado con el análisis de la Tabla A-2 y el ANEXO III; como se puede ver, la imagen 28 no se ve favorecida con este procesamiento, sin embargo al comparar estos resultados con los resultados obtenidos sin realizar ninguna modificación a la imagen ni al OCR, los resultados obtenidos son bastante significativos.

En la siguiente tabla se resumen los procesamientos comunes encontrados con base en las características de las fuentes analizadas en el ANEXO III.

Estilo Fuente	Preprocesado
Negrilla/CursivaNegrilla	Bicubic 3X
Roman	Bilinear 2X
Cursiva	Bicubic 3X-2X

Tabla A-3: Procesamiento común encontrado en función del estilo de fuente

En la Ilustración A-9, se muestran los resultados obtenidos al aplicar el procesamiento común según la Tabla A-1 para las imágenes cuyo estilo de fuente arrojado por el OCR es Negrilla y CursivaNegrilla; en la Ilustración A-10 se grafican los resultados para el estilo de fuente Roman y en la Ilustración A-11 el estilo de fuente cursiva.

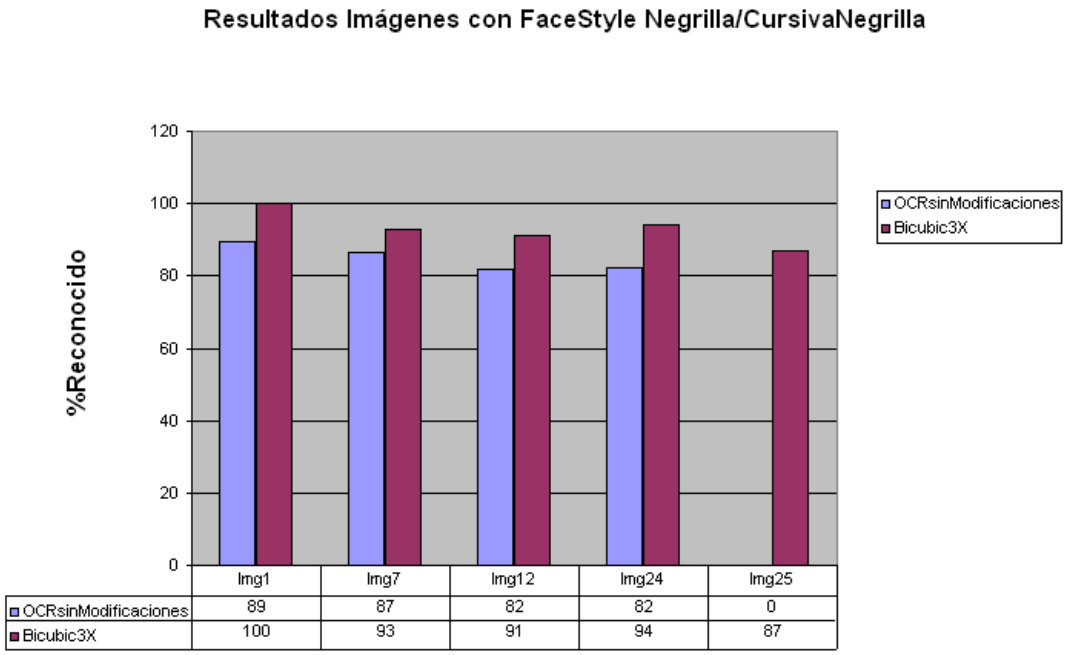


Ilustración A-9: Resultados al aplicar un remuestreo Bicúbico 3X para las imágenes cuyo estilo de fuente es Negrilla y CursivaNegrilla

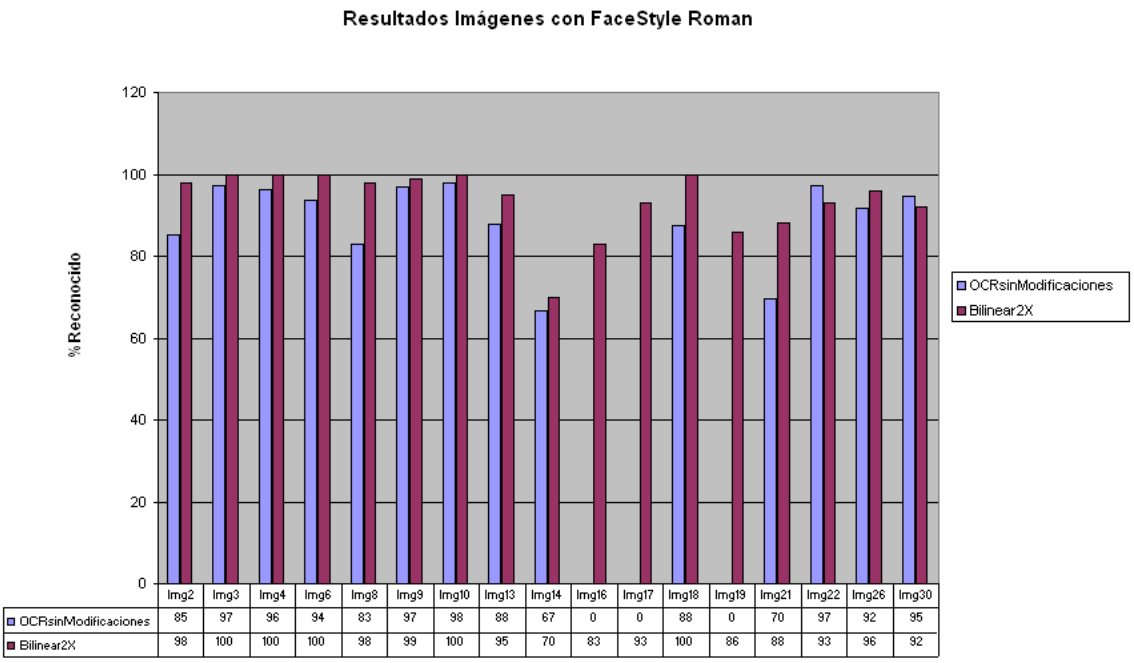


Ilustración A-10: Resultados al aplicar un remuestreo Bilinear 2X para las imágenes cuyo estilo de fuente es Roman

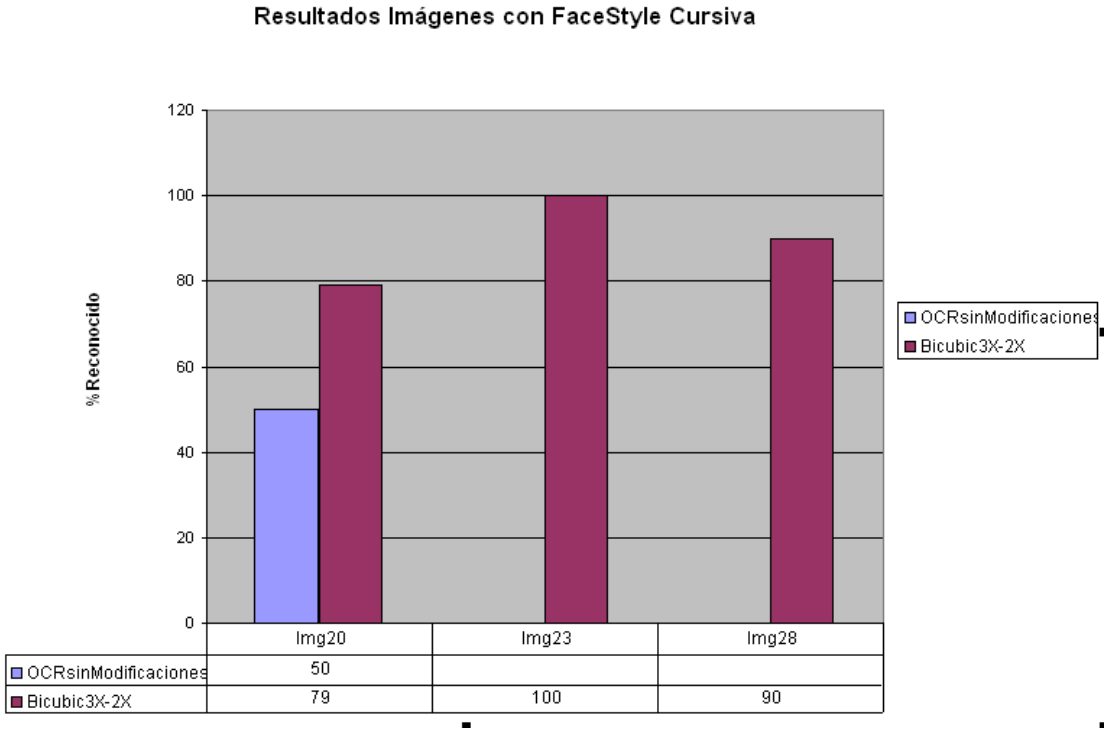
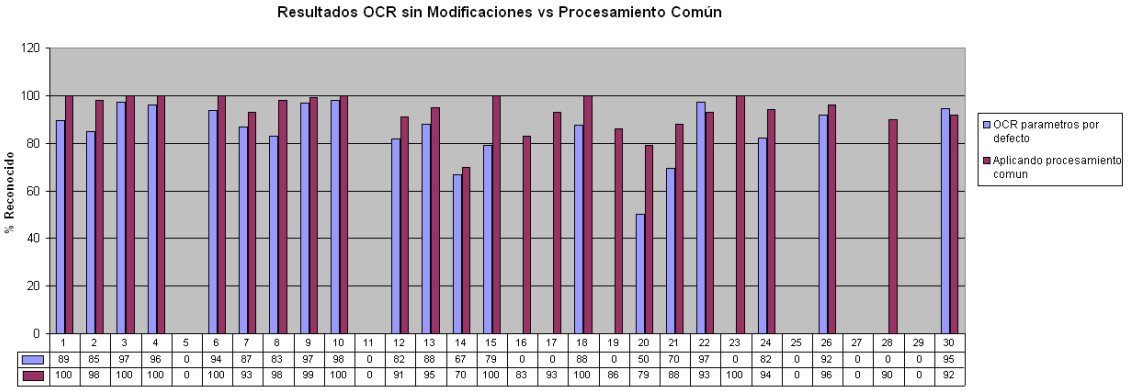


Ilustración A-11: Resultados al aplicar un remuestreo Bicubic 3X-2X para las imágenes cuyo estilo de fuente es Cursiva

Como se puede observar en las figuras anteriores, la generalización ha sido bastante buena lográndose mejoras significativas en el porcentaje de letras reconocidas en las imágenes.

En la Ilustración A-12 (ANEXO IX) se exponen en conjunto los resultados obtenidos en el reconocimiento al ejecutar un preprocesado común para todas las imágenes basado en el FaceStyle de la fuente (barras color púrpura), comparado con los resultados obtenidos al aplicar el OCR a las imágenes sin ningún tipo de procesamiento (barras celestes).

Como se aprecia claramente, al comparar estos valores con los obtenidos inicialmente, se logran mejoras bastante significativas en el porcentaje de acierto conservándose un valor alto para todo el conjunto de imágenes, pasando de una tasa de reconocimiento alrededor del 69% al 91%. En estos resultados no se incluye la información de las imágenes (5, 11, 27 y 29) ya que dichas imágenes fueron las que menor tasa de reconocimiento arrojaron en las diferentes pruebas y teniendo en cuenta el objetivo de esta segunda etapa, era necesario trabajar con las imágenes que más información pudieran aportar a los resultados de las pruebas.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
	89	85	97	96	0	94	87	83	97	98	0	82	88	67	79	0	0	88	0	50	70	97	0	82	0	92	0	0	0	95
	100	98	100	100	0	100	93	98	99	100	0	91	95	70	100	83	93	100	86	79	88	93	100	94	0	96	0	90	0	92

Ilustración A-12: Resultados al ejecutar el OCR sin ninguna modificación comparado con los resultados obtenidos al aplicar un procesamiento común basado en el FaceStyle

Los resultados obtenidos en estas dos etapas arrojaron información importante sobre la posibilidad de mejorar la tasa de aciertos del OCR al aplicar un preprocesado a las imágenes. Esta mejora se logra a través del algoritmo descrito en la Ilustración A-13.

La primera parte del algoritmo consiste en asegurar que se lleve a cabo un reconocimiento en la imagen; como se concluyó en la etapa I, esto se puede lograr a través de remuestreos

progresivo 2X, 3X … 8X, con los diferentes tipos de interpolación (bicúbica, bilineal y por similitud) hasta que se logre un reconocimiento de caracteres en la imagen.

Una vez logrado un primer reconocimiento, es posible extraer información de la imagen reconocida; como su tipo de fuente, tamaño en píxeles y estilo, entre otros; con el fin de usar esta información para aplicar un tipo de procesamiento basado en dichas características; este segundo procesamiento se basará en los resultados obtenidos en la etapa II donde se encontró que para un FaceStyle negrilla y CursivaNegrilla, un remuestreo bicúbico de 3X arroja los mejores resultados; para un FaceStyle Roman un remuestreo bilineal de 2X y para un FaceStyle Cursiva un remuestreo bicúbico 3X a lo alto y 2X a lo ancho.

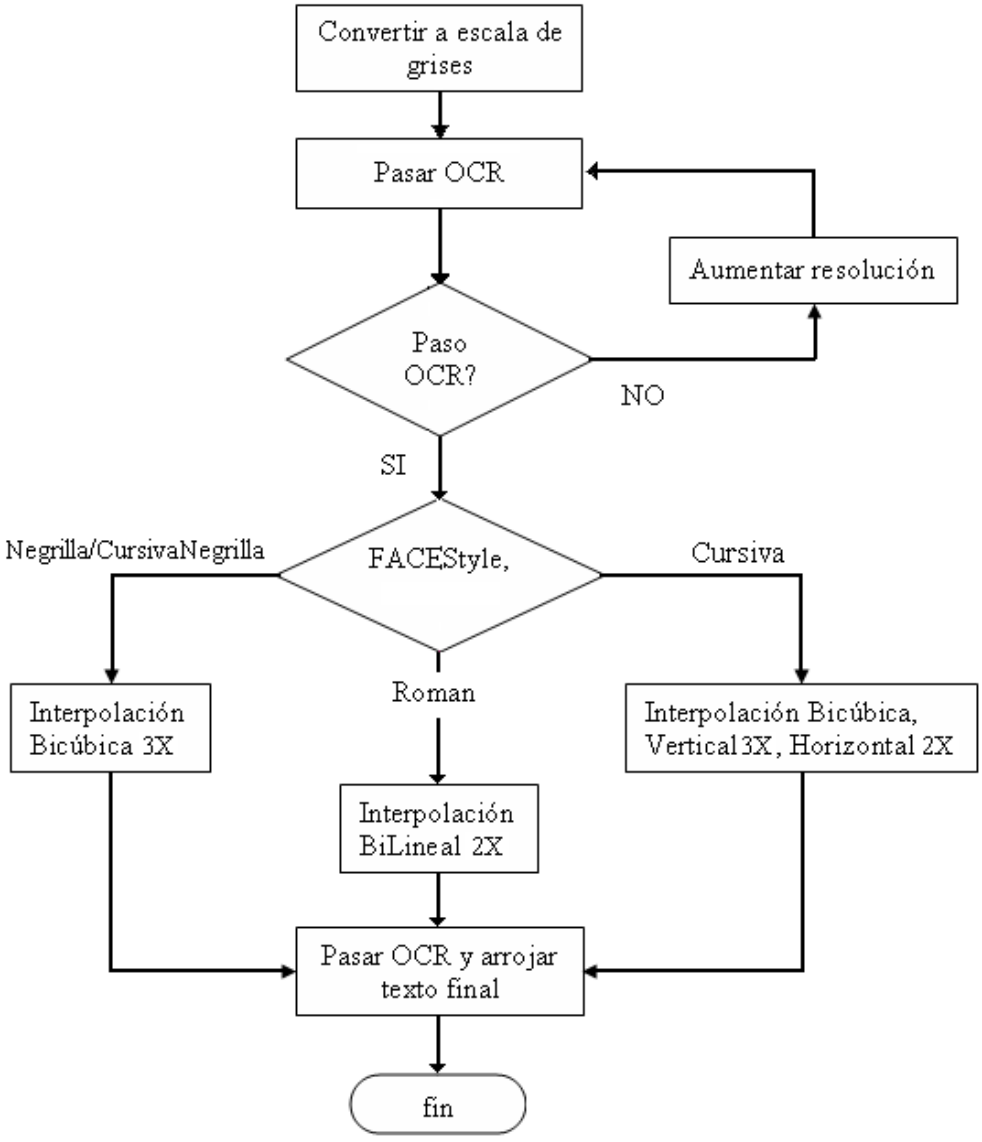


Ilustración A-13: Diagrama de flujo, algoritmo para la mejora del OCR

Estudio comparativo de Paquetes de Software de Reconocimiento de caracteres

Para generar alternativas al software de reconocimiento de caracteres de Microsoft MODI, se ha realizado una búsqueda exhaustiva de diferente software de OCR que oferta el mercado. Dicha búsqueda arrojó como resultado una serie de material que se puede clasificar en dos grandes grupos, uno que abarca los productos comerciales de empresas que se dedican única y exclusivamente al desarrollo de software de reconocimiento óptico de caracteres y otro que se compone de iniciativas de desarrollo de código libre.

Aunque existe gran variedad de software disponible en Internet, la búsqueda se concentro en aquellos que permitirían mediante un SDK o un API, integrar su funcionalidad en otras aplicaciones. La lista base de búsqueda, es la publicada en la Wikipedia [1]. Partiendo de esta lista se subdivide el estudio en las dos categorías ya mencionadas. A continuación se listan divididos en dichas categorías.

Software Comercial

ABBYY FineReader OCR: Software desarrollado

Adobe Acrobat:

NovoDynamics VERUS:

ReadSoft:

SimpleOCR:

Software libre

Tesseract:

Ocrad:

GOCR:

A.1.2 Pruebas realizadas

Para la realización de las pruebas se utilizó la base de datos de 30 imágenes mostradas en el ANEXO I. Estas imágenes no han sido modificadas con los procedimientos de remuestreo, puesto que el objetivo de esta prueba es hacer una selección del software que posee un mejor desempeño *per se* ante una variedad de imágenes con características

heterogéneas como lo serán las que se extraigan de la red, que serán la entrada usual para el reconocedor de caracteres.

Una vez realizado el contacto con todas la empresas y probado el software listado anteriormente, la primera revisión de los resultados obtenidos descartó algunas de las alternativas por lo cual no se incluyen en las tablas resumen que se muestran a continuación. En las tablas Tabla A-4 y Tabla A-5 se incluyen los porcentajes de acierto promedio para el software comercial y libre, con los programas que obtuvieron mejores resultados; en los Anexos ANEXO IV y ANEXO V se encuentran los datos de las pruebas realizadas. Además las ilustraciones Ilustración A-14 y Ilustración A-15 (Anexos ANEXO X y ANEXO XI) muestran los resultas obtenidos para cada una de las imágenes con los distintos software testeados.

Imágenes	Software Comercial		
	NovoDynamics Verus	ABBYY FineReader 9,0	MODI
	% Reconocido Caracteres	% Reconocido Caracteres	% Reconocido Caracteres
Promedio	**74,93**	**93,58**	**57,24**

Tabla A-4: Resultados obtenidos con el software comercial

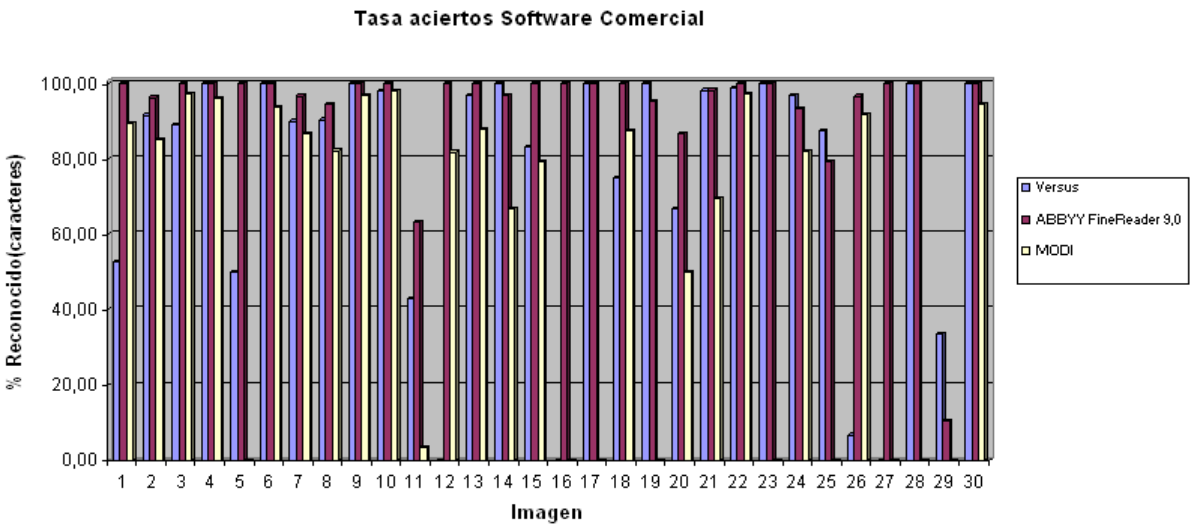


Ilustración A-14: Resultados pruebas realizadas con software comercial

Imágenes	Software Libre					
	GOCR		Tesseract		OCRAD	
	% Reconocido Caracteres	%Reconocido Palabras	% Reconocido Caracteres	%Reconocido Palabras	% Reconocido Caracteres	%Reconocido Palabras
Promedio	**58,30**	**34,14**	**44,78**	**20,70**	**60,10**	**28,37**

Tabla A-5: Resultados obtenidos con el software libre

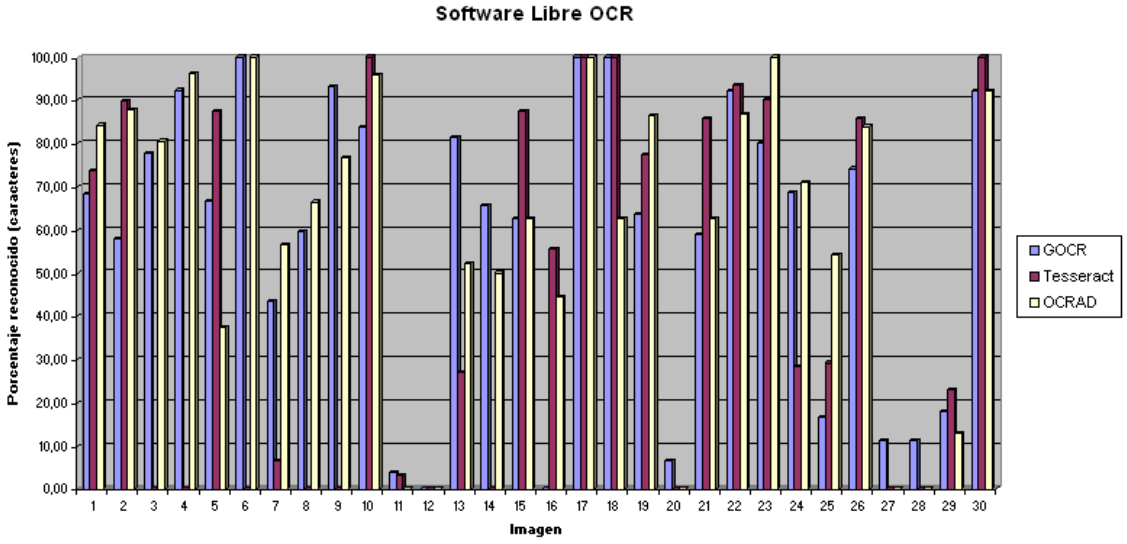


Ilustración A-15: Resultados pruebas realizadas con software libre

Analizando los resultados de las tablas Tabla A-4 y Tabla A-5, es evidente que el software comercial obtuvo un mejor desempeño en lo que se refiere al reconocimiento de caracteres que los de código libre. El mejor software de reconocimiento resultó ser el ABBY FineReader, que alcanzó un consistente **93,58%** de aciertos en su tasa promedio, incluyendo las deterioradas imágenes 11 y 29.

Es importante resaltar que las condiciones de las imágenes que sirvieron de entrada, son las que originalmente se bajaron de la red sin ningún tipo de procesamiento. Se realizaron pruebas adicionales sobre el comportamiento del ABBY FineReader frente a diferentes modificaciones en la imágenes con el fin de mejorar el reconocimiento en aquellas imágenes que presentaron problemas significativos (concretamente las imágenes 11, 20, 25 y 29); estas pruebas arrojaron mejoras porcentuales en 3 de las 4 imágenes que estaban en el rango del 10% al 30%.

Los procedimientos aplicados a las imágenes consistieron en homogenizarlas a una resolución de 300ppi. El resultado fue satisfactorio, pues mejoraron en el reconocimiento las imágenes deterioradas mencionadas anteriormente. La normalización de la resolución obtuvo resultados adecuados para una aplicación automatizada de reconocimiento en el 86% de las imágenes usadas para la prueba con una tasa de acierto promedio de 93.58%. Cabe anotar que la resolución usada de 300ppi se eligió para asegura las condiciones de funcionamiento estándar de los diferentes paquetes, sin embargo como se muestra en la Ilustración A-13, el uso del aumento de la resolución en forma selectiva produjo mejores resultados.

Conclusiones y recomendaciones

- Como resultado del estudio comparativo de los diferentes paquetes de software que proveen un API o SDK para integrar su funcionalidad a una aplicación queda claro y contundentemente soportado por las pruebas, que la primera medida es cambiar del MODI al SDK de ABBYY. Las pruebas fueron efectuadas con el front-end llamado FineReader, el ABBY FineReader Engine es el SDK de este producto.
- El estudio sobre los diferentes tipos de procesamiento sobre la imágenes y la relación que existe con características de las mismas que permitan automatizar el proceso, permitieron estructurar un algoritmo de procesamiento (ver Ilustración A-10) y precisar características de la imagen para mejorar la calidad de los

resultados del OCR (en particular del MODI, pero perfectamente extrapolable a otros). Dichos procedimientos se describen a continuación:

- Convertir las imágenes a escala de grises según el canal de luminosidad.
- Pasar el OCR; si en primera instancia no se logra que el OCR reconozca ningún tipo de texto, se debe proceder a aumentar la resolución al doble de la original usando la interpolación bicúbica. Cada vez que el OCR no logre pasar se debe proceder a incrementar en 1 el factor de multiplicación de la resolución respecto de la resolución original hasta un máximo de 6 veces y si es el caso, cambiar igualmente de tipo de interpolación (bilineal y por similitud) hasta conseguir un reconocimiento. Por ejemplo si una imagen no logra ser reconocida al duplicar la resolución, entonces se debe intentar con la resolución original multiplicada por 3.
- Una vez se logre de manera exitosa reconocer algún tipo de texto, los OCR arrojan como resultado de dicho reconocimiento parámetros como el FaceStyle de la letra y el tamaño en píxeles. Esta información es usada para hacer un nuevo procesamiento a la imagen. Este nuevo procesamiento depende de las características extraídas de la siguiente forma:
  - Si el FaceStyle es Negrilla o CursivaNegrilla, se debe hacer una interpolación bicúbica de la imagen al triple de la resolución actual.
  - Si el FaceStyle es Roman se debe hacer una interpolación bilineal al doble de la resolución actual.
  - el FaceStyle es Cursiva se debe hacer una interpolación bicúbica al triple de la resolución vertical y al doble en la horizontal.
- Una vez realizado el procesamiento se pasa de nuevo el OCR y como resultado final se obtiene el texto de la imagen.

- El algoritmo propuesto ha sido desarrollado con imágenes a resolución estándar Web (de 72 a 96 dpi); si es necesario trabajar con imágenes a otra resolución, el remuestreo de las imágenes propuesto en el algoritmo puede ser modificado para usar rangos de resolución en vez de factores de escala.
- El algoritmo propuesto se basa en la información que da el MODI, sin embargo basados en las posibilidades que ofrece el front-end de ABBY, se puede deducir que algunas mejoras se podrían introducir gracias a que hay más información sobre la imagen ligada al reconocimiento de texto que proporciona este SDK.

- En vista de la división existente en un punto del algoritmo en tres tipos diferentes de procesamiento para las imágenes y sujetos a la alta probabilidad de encontrarnos con imágenes que manejan diferentes tipos de letra (lo cual dificulta aplicar el procesamiento adecuando), es posible pensar en enlazar a la misma imagen procesada dos resultados distintos. Esta decisión permite al usuario de manera transparente hallar el texto que mejor se ajusta a su búsqueda. Los dos resultados propuestos corresponderían a las dos veces en la que el OCR es ejecutado. Esta ventaja deberá ser contrarrestada con la rapidez del algoritmo y el tamaño de la base de datos las cuales se pueden ver bastante afectadas si el análisis de diagramas se hace de manera masiva.
- Teniendo en cuenta que las pruebas y las conclusiones obtenidas en este estudio son basadas en un conjunto de 30 imágenes, cabría la posibilidad de extender las pruebas a un conjunto de imágenes mayor, con el fin de analizar más a fondo la robustez del algoritmo propuesto. En este caso vale la pena resaltar que debido a la forma como el OCR segmenta las palabras, no es posible automatizar la tarea de comparación de los resultados obtenidos con la información presente en la imagen haciendo que dicha labor se deba realizar de forma manual.

Recomendaciones Finales

- Debido a que un trozo de imagen puede contener distintos tipos de letra podría ser recomienda poder segmentar la imagen según el tipo de letra y procesar las palabras con el tipo y tamaño de letra usando el algoritmo propuesto.
- El eliminar cualquier tipo de subrayado en las imágenes mejora los resultados del reconocedor.
- Estudiar y emplear a fondo las capacidades que provee el API del FineReader Engine que incluye también herramientas de filtrado y retención del layout para mejorar el algoritmo. Este motor de reconocimiento arroja el grado de certidumbre con el que se reconoce un carácter y la correspondiente región en la imagen que representa dicho carácter. Así, es posible pensar en crear subrutinas que hagan preprocesado sobre esa región de la imagen en concreto e identificar el carácter; adicionalmente esta información puede ser usada para tener una estimación del error cometido en el reconocimiento basado en la cantidad de regiones y la cantidad de caracteres cuyo grado de certidumbre es suficientemente alto como para considerarse confiable. Sin embrago cabe anotar que esta sugerencia se hace

sin un conocimiento profundo de las funcionalidades del SDK, y que estaría sujeta a fases posteriores de desarrollo.

ANEXO I

<<Interface>>
Figure

public float perimeter()
public void rotate(Point center, float angle)
public translate(float x, float y)
public display()

«Code»
20022 Test Data Types::TestCode

tags
maxLength = 2
minLength = 2

(from Formal Requirements)

Name
Company
Address
Telephone
Email

<<Interface>>
ContextMembership

accountName : string
nickname : string
fullName : string
emailAddress : string
organization : string
phoneNumber : string
postalAddress : string
fax : string
image : string

- id: String
- name: String
- address: String
- zip: String
- city: String
- state: String

ID, Status, Priority, Time,
Communicator, Issue Content

+Serial_Num : Long
+RAM : Long
+CPU : String
+Gen_Location : String
+Type : String
+Manufacturer : String
+Manufact_Date : Date
+Examiner_Comments : String
+User_Tags : String
+Num_Internal_Drives : Short

Application

-companyName: CString
-experience: CString
-reference1: CString
-reference2: CString
-reference3: CString

+Application ()
+ApplicationInfo ()

<< abstract >>
AbstractCell

+$colors : Hashtable

+ static classAttribute: int

BodyCell

+handleClick(mode : int)

+ *void : doStuff()*

WEBBROWSER_DOWNLOAD_STARTED
WEBBROWSER_DOWNLOAD_COMPLETED

public int addDoctor()
public int addStaff()
public void delDoctor(int id)
public void delStaff(int id)
public void editDoc(int id)
public void editStaff(int id)

name: String [0..1]
visibility: VisibilityKind [0..1]
/ qualifiedName: String [0..1]

Classifier

+ ABall(Point p, int r, Point v, Color c, Container container)
+ void : update(Observable o, Object g)
+ void : bounce()
+ void : setLocation(Point location)
+ Point : getLocation()
+ void : setRadius(int radius)
+ int : getRadius()
+ void : setVelocity(Point velocity)
+ Point : getVelocity()
+ void : setColor(Color color)
+ Color : getColor()
+ void : paint(Graphics g)
+ *void : updateState()*

+ *Object : apply(Object arg)*

Character

AComponent

<<FreeRoleType>>
FigureObserver /Observer

+ static instanceVariable : SingletonClass

ANEXO II

Imagen	Prueba1_OCR sin modificaciones		Prueba2_OCR modificandoLanguage		Prueba3_CANAL_R(0)		Prueba4_CANAL_G(1)		Prueba5_CANAL_B(2)		Prueba6_GRAY	
	Letras		Letras		Letras		Letras		Letras		Letras	
	ACIERTOS	% ACIERTO	ACIERTOS	% ACIERTO	ACIERTOS	% ACIERTO	ACIERTOS	% ACIERTO	ACIERTOS	% ACIERTO	ACIERTOS	% ACIERTO
1	17	89	18	95	16	84	16	84	16	84	16	84
2	91	85	95	89	98	92	98	92	98	92	98	92
3	35	97	31	86	32	89	32	89	31	86	31	86
4	25	96	25	96	23	88	25	96	24	92	26	100
5	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0
6	30	94	32	100	32	100	32	100	32	100	32	100
7	26	87	26	87	27	90	27	90	27	90	27	90
8	120	83	127	87	130	89	130	89	129	88	126	86
9	65	97	65	97	67	100	66	99	67	100	65	97
10	48	98	48	98	49	100	49	100	49	100	49	100
11	5	3	5	3	40	25	40	25	40	25	40	25
12	9	82	11	100	11	100	11	100	11	100	11	100
13	80	88	82	90	84	92	84	92	82	90	84	92
14	20	67	22	73	20	67	20	67	20	67	19	63
15	19	79	19	79	22	92	22	92	22	92	22	92
16	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0

17	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0
18	7	88	8	100	8	100	8	100	8	100	8	100
19	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0
20	7	43	8	57	10	71	10	71	10	71	9	64
21	39	70	39	70	41	73	41	73	41	73	41	73
22	73	97	74	99	71	95	71	95	71	95	71	95
23	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0
24	110	82	113	84	117	87	117	87	117	87	123	92
25	0	43	0	0	18	78	18	78	18	78	16	70
26	302	92	277	84	299	91	299	91	299	91	299	91
27	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0
28	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0
29	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0
30	35	95	34	92	37	100	37	100	37	100	37	100

Imagen	Prueba7_Satur		Prueba8_Value		Prueba9_2X		Prueba10_4X		Prueba11_6X		Prueba12_300ppi	
	Letras		Letras		Letras		Letras		Letras		Letras	
	ACIERTOS	% ACIERTO	ACIERTOS	% ACIERTO	ACIERTOS	% ACIERTO	ACIERTOS	% ACIERTO	ACIERTOS	% ACIERTO	ACIERTOS	% ACIERTO
1	NoOCR	0	16	84	18	95	18	94,7	18	94,7	18	95
2	NoOCR	0	98	92	95	89	101	94,4	103	96,3	95	89
3	21	58	31	86	31	86	32	88,9	32	88,9	31	86
4	20	77	24	92	25	96	25	96,2	24	92,3	25	96
5	NoOCR	0	NoOCR	0	NoOCR	0	16	66,7	16	66,7	NoOCR	0
6	32	100	32	100	32	100	32	100,0	32	100,0	32	100
7	21	70	27	90	26	87	29	96,7	28	93,3	26	87
8	101	69	130	89	126	86	136	93,2	142	97,3	126	86
9	66	99	67	100	66	99	63	94,5	61	91,8	66	99
10	0	0	49	100	48	98	49	100,0	49	100,0	48	98
11	NoOCR	0	40	25	11	7	8	5,1	4	2,5	11	7
12	11	100	11	100	11	100	2	18,2	0	0,0	11	100
13	NoOCR	0	82	90	82	90	86	94,8	82	90,6	82	90
14	NoOCR	0	20	67	22	73	29	96,9	23	78,1	22	73
15	NoOCR	0	22	92	19	79	23	95,8	22	91,7	19	79
16	NoOCR	0	NoOCR	0	NoOCR	0	14	77,8	16	88,9	NoOCR	0
17	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0	22	92,0	NoOCR	0

18	NoOCR	0	8	100	8	100	7	87,5	7	87,5	8	100
19	NoOCR	0	NoOCR	0	NoOCR	0	21	100,0	15	72,7	NoOCR	0
20	2	14	NoOCR	0	8	57	13	93,3	13	95,6	8	57
21	NoOCR	0	41	73	41	73	56	100,0	51	91,1	41	73
22	NoOCR	0	71	95	74	99	75	100,0	73	97,3	74	99
23	NoOCR	0	NoOCR	0	NoOCR	0	10	100,0	10	100,0	NoOCR	0
24	118	88	117	87	109	81	123	91,8	123	91,8	109	81
25	NoOCR	0	16	70	8	35	19	83,3	19	83,3	8	35
26	NoOCR	0	299	91	288	88	323	98,2	319	97,0	288	88
27	NoOCR	0	NoOCR	0	NoOCR	0	8	84,7	8	87,5	NoOCR	0
28	NoOCR	0	NoOCR	0	NoOCR	0	0	0	0	0	NoOCR	0
29	NoOCR	0	NoOCR	0	NoOCR	0	20	51,3	21	53,8	NoOCR	0
30	NoOCR	0	37	100	35	95	0	0	0	0	35	95

Imagen	Prueba13_300ppiRojo		Prueba14_300ppiRojoErod1		Prueba15_300ppiRojoErod2		Prueba16_300ppiRojoDila1		Prueba17_300ppiRojoDila2	
	Letras		Letras		Letras		Letras		Letras	
	ACIERTOS	% ACIERTO	ACIERTOS	% ACIERTO	ACIERTOS	% ACIERTO	ACIERTOS	% ACIERTO	ACIERTOS	% ACIERTO
1	17	89	14	74	14	74	17	89	17	89
2	96	90	84	79	84	79	94	88	77	72
3	31	86	32	89	32	89	32	89	29	81
4	26	100	25	96	25	96	25	96	9	35
5	16	67	14	58	14	58	13	54	4	17
6	7	22	10	31	10	31	0	0	4	13
7	14	47	26	87	26	87	26	87	27	90
8	117	80	113	77	113	77	106	73	100	68
9	58	87	55	82	55	82	53	79	46	69
10	0	0	47	96	47	96	48	98	48	98
11	0	0	10	6	10	6	6	4	0	0
12	6	55	8	73	8	73	6	55	0	0
13	79	87	77	85	77	85	69	76	73	80
14	22	73	20	67	20	67	22	73	0	0
15	20	83	19	79	19	79	19	79	22	92
16	8	44	4	22	4	22	NoOCR	0	7	39
17	22	92	23	96	23	96	21	88	21	88

18	7	88	8	100	8	100	8	100	8	100
19	13	62	15	71	15	71	9	43	NoOCR	0
20	10	71	8	57	8	57	9	64	8	57
21	44	79	32	57	32	57	49	88	29	52
22	67	89	65	87	65	87	65	87	49	65
23	0	0	NoOCR	0	NoOCR	0	10	100	10	100
24	105	78	106	79	106	79	97	72	74	55
25	15	65	9	39	9	39	11	48	10	43
26	241	73	227	69	230	70	251	76	221	67
27	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0	NoOCR	0
28	10	100	6	60	5	50	10	100	10	100
29	0	0	6	15	0	0	0	0	10	26
30	36	97	36	97	36	97	35	95	28	76

ANEXO III

Imágenes	Porcentaje	Preprocesado		FaceStyle	Tamaño fuente Inicial	Observaciones
		Canal	Remuestreo			
Img1	100%	Gris	Bicubic 3X	Roman	9	
Img2	98%	Gris	Bilinear 2X	Roman	8	
Img3	100%	Gris	Bilinear 2X	Roman	8	
Img4	100%	Gris	Bilinear 2X	Roman	8	
Img5		Gris				
Img6	100%	Gris	Bilinear 2X	Roman	11	sin remuestreo se logra un 100%
Img7	93%	Gris	Bicubic 3X	Negrilla	8	
Img8	98%	Gris	Bilinear2X	Negrilla	8	
Img9	100%	Gris	Bicubic 2X	Roman	8	
Img10	100%	Gris	Bilinear 2X	Roman	8	
Img11		Gris				
Img12	100%	Gris	Bilinear 3X	Roman	11	sin remuestreo se logra un 100%
Img13	99%	Gris	Bicubic 2X	Roman	9	

Img14	97%	Gris	Bicubic 3X	Roman	11	
Img15	100%	Gris	Bicubic 2X	CursivaNegrilla/Roman	8	
Img16	94%	Gris	Bicubic 3X + Dilate	Roman	8	
Img17	96%	Gris	Bilinear 2X	Roman	9	
Img18	100%	Gris	Bilinear 2X	Roman	9	sin remuestreo se logra un 100%
Img19	95%	Gris	Bicubic 3X	Roman		
Img20	79%	Gris	Bicubic 3X-2X	Negrilla	10	
Img21	96%	Gris	Nearest 2X	Roman	9	
Img22	100%	Gris	Bicubic 3X	Roman	9	
Img23	100%	Gris	Bicubic 3X-2X	Cursiva		
Img24	94%	Gris	Bicubic 3X	Roman	8	
Img25	91%	Gris	Bicubic3X+Dilate	CursivaNegrilla	10	
Img26	96%	Gris	Bilinear2X	Roman	10	
Img27		Gris				
Img28	90%	Gris	Bicubic 3X-2X	CursivaNegrilla		
Img29		Gris				
Img30	100%	Gris	Bicubic3X	Roman	8	

ANEXO IV

Imagen	Estilo Fuente	Resultado OCR parámetros por defecto	Resultado OCR con procesamiento basado en el FaceStyle	Procesamiento común
Img1	Roman	89%	84%	Bilinear 2X
Img2	Roman	85%	98%	Bilinear 2X
Img3	Roman	97%	100%	Bilinear 2X
Img4	Roman	96%	100%	Bilinear 2X
Img5		NoOCR		
Img6	Roman	94%	100%	Bilinear 2X
Img7	Negrilla	87%	93%	Bicubic 3X
Img8	Negrilla	83%	92%	Bicubic 3X
Img9	Roman	97%	99%	Bilinear 2X
Img10	Roman	98%	100%	Bilinear 2X
Img11		3%		
Img12	Roman	82%	91%	Bilinear 2X
Img13	Roman	88%	95%	Bilinear 2X
Img14	Roman	67%	70%	Bilinear 2X
Img15	CursivaNegrilla/Roman	79%	88%	Bicubic 3X
Img16	Roman	NoOCR	83%	Bilinear 2X
Img17	Roman	NoOCR	93%	Bilinear 2X

Img18	Roman	88%	100%	Bilinear 2X
Img19	Roman	NoOCR	86%	Bilinear 2X
Img20	Negrilla	43%	43%	Bicubic 3X
Img21	Roman	70%	88%	Bilinear 2X
Img22	Roman	97%	93%	Bilinear 2X
Img23	Cursiva	NoOCR	100%	Bicubic 3X-2X
Img24	Roman	82%	92%	Bilinear 2X
Img25	CursivaNegrilla	43%	87%	Bicubic 3X
Img26	Roman	92%	96%	Bilinear 2X
Img27		NoOCR		
Img28	CursivaNegrilla	NoOCR	90%	Bicubic 3X
Img29		NoOCR		
Img30	Roman	95%	92%	Bilinear 2X

ANEXO V

Imagen	Nº Total Caracteres (sin espacios)	Nº Total Palabras	Software Comercial							
			Versus				ABBYY FineReader 9,0			
			Caracteres Reconocidos	% Reconocido Caracteres	Palabras Reconocidas	%Reconocido Palabras	Caracteres Reconocidos	% Reconocido Caracteres	Palabras Reconocidas	%Reconocido Palabras
Img1	19	2	10	52,63	0	0,00	19	100,00	2	100,00
Img2	107	16	98	91,59	13	81,25	103	96,26	16	100,00
Img3	36	5	32	88,89	4	80,00	36	100,00	4	80,00
Img4	26	7	26	100,00	7	100,00	26	100,00	7	100,00
Img5	24	3	12	50,00	0	0,00	24	100,00	3	100,00
Img6	32	5	32	100,00	5	100,00	32	100,00	5	100,00
Img7	30	2	27	90,00	0	0,00	29	96,67	1	50,00
Img8	146	27	132	90,41	17	62,96	138	94,52	23	85,19
Img9	73	18	73	100,00	18	100,00	73	100,00	18	100,00

Img10	49	7	48	97,96	6	85,71	49	100,00	7	100,00
Img11	158	26	68	43,04	14	53,85	100	63,29	16	61,54
Img12	11	1	0	0,00	0	0,00	11	100,00	1	100,00
Img13	96	10	93	96,88	10	100,00	96	100,00	10	100,00
Img14	32	4	32	100,00	3	75,00	31	96,88	4	100,00
Img15	24	2	20	83,33	1	50,00	24	100,00	2	100,00
Img16	18	2	0	0,00	0	0,00	18	100,00	2	100,00
Img17	25	4	25	100,00	3	75,00	25	100,00	4	100,00
Img18	8	1	6	75,00	0	0,00	8	100,00	1	100,00
Img19	22	2	22	100,00	1	50,00	21	95,45	1	50,00
Img20	15	4	10	66,67	3	75,00	13	86,67	3	75,00
Img21	56	2	55	98,21	1	50,00	55	98,21	1	50,00
Img22	75	13	74	98,67	12	92,31	75	100,00	13	100,00
Img23	10	1	10	100,00	1	100,00	10	100,00	1	100,00

Img24	134	25	130	97,01	21	84,00	125	93,28	20	80,00
Img25	24	4	21	87,50	2	50,00	19	79,17	1	25,00
Img26	329	59	22	6,69	11	18,64	318	96,66	55	93,22
Img27	9	1	0	0,00	0	0,00	9	100,00	1	100,00
Img28	9	1	9	100,00	1	100,00	9	100,00	1	100,00
Img29	39	7	13	33,33	1	14,29	4	10,26	0	0,00
Img30	38	4	38	100,00	3	75,00	38	100,00	3	75,00

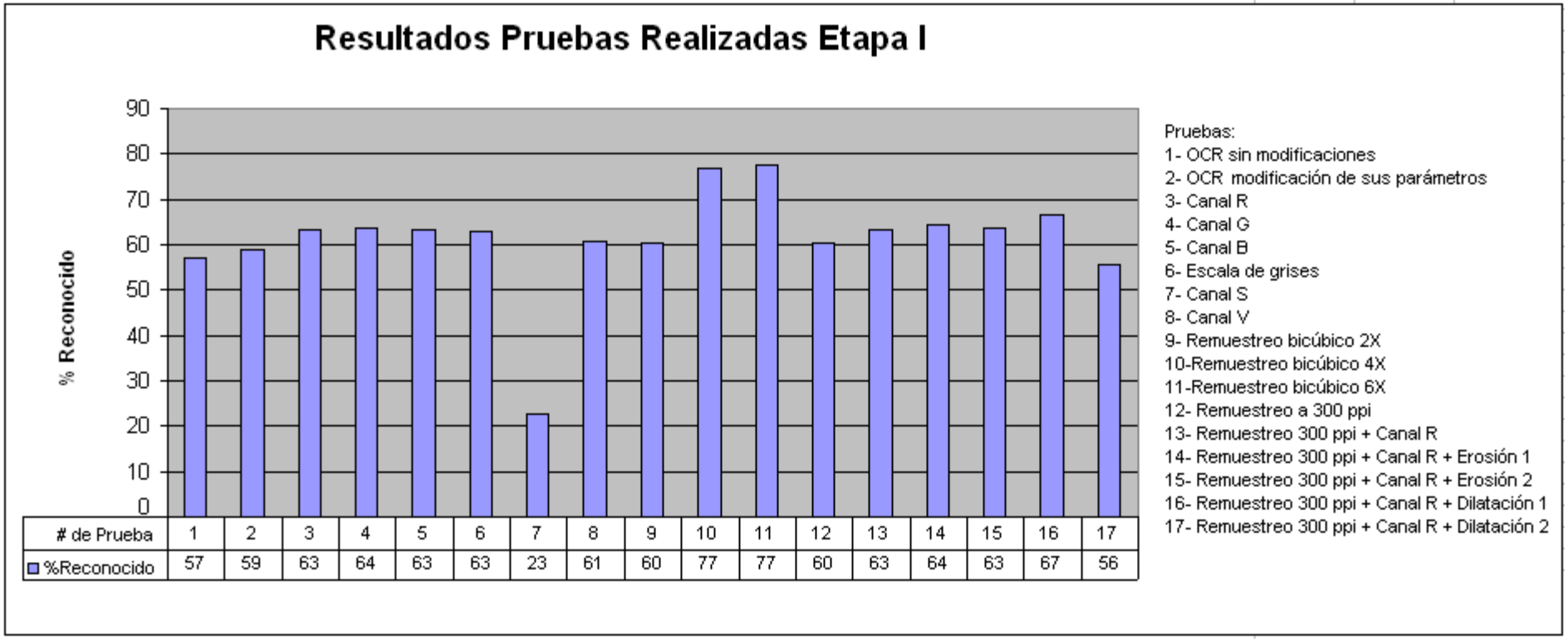
ANEXO VI

Imagen	Software Libre											
	GOCR				TESSERACT				OCRAD			
	Caracteres Reconocidos	% Reconocido Caracteres	Palabras Reconocidas	%Reconocido Palabras	Caracteres Reconocidos	% Reconocido Caracteres	Palabras Reconocidas	%Reconocido Palabras	Caracteres Reconocidos	% Reconocido Caracteres	Palabras Reconocidas	%Reconocido Palabras
Img1	13	68,42	0	0,00	14	73,68	0	0,00	16	84,21	0	0,00
Img2	62	57,94	7	43,75	96	89,72	11	68,75	94	87,85	8	50,00
Img3	28	77,78	3	60,00	0	0,00	0	0,00	29	80,56	2	40,00

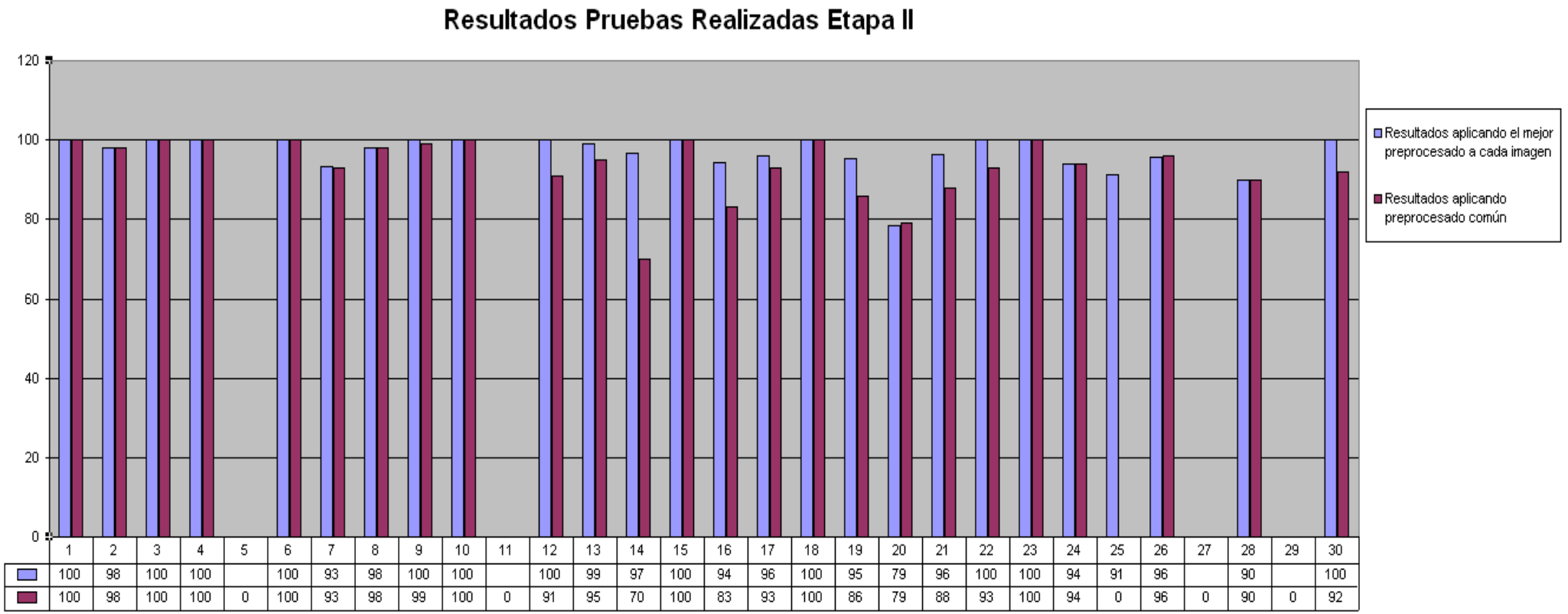
Img4	24	92,31	5	71,43	0	0,00	0	0,00	25	96,15	6	85,71
Img5	16	66,67	0	0,00	21	87,50	1	33,33	9	37,50	0	0,00
Img6	32	100,00	5	100,00	0	0,00	0	0,00	32	100,00	5	100,00
Img7	13	43,33	0	0,00	2	6,67	0	0,00	17	56,67	0	0,00
Img8	87	59,59	11	40,74	0	0,00	0	0,00	97	66,44	10	37,04
Img9	68	93,15	15	83,33	0	0,00	0	0,00	56	76,71	8	44,44
Img10	41	83,67	4	57,14	49	100,00	7	100,00	47	95,92	5	71,43
Img11	6	3,80	0	0,00	5	3,16	0	0,00	0	0,00	0	0,00
Img12	0	0,00	0	0,00	0	0,00	0	0,00	0	0,00	0	0,00
Img13	78	81,25	1	10,00	26	27,08	0	0,00	50	52,08	0	0,00
Img14	21	65,63	2	50,00	0	0,00	0	0,00	16	50,00	0	0,00
Img15	15	62,50	0	0,00	21	87,50	0	0,00	15	62,50	0	0,00
Img16	0	0,00	0	0,00	10	55,56	1	50,00	8	44,44	0	0,00
Img17	25	100,00	4	100,00	25	100,00	4	100,00	25	100,00	4	100,00
Img18	8	100,00	1	100,00	8	100,00	1	100,00	5	62,50	0	0,00
Img19	14	63,64	1	50,00	17	77,27	0	0,00	19	86,36	1	50,00

Img20	1	6,67	0	0,00	0	0,00	0	0,00	0	0,00	0	0,00
Img21	33	58,93	0	0,00	48	85,71	0	0,00	35	62,50	0	0,00
Img22	69	92,00	10	76,92	70	93,33	9	69,23	65	86,67	5	38,46
Img23	8	80,00	0	0,00	9	90,00	0	0,00	10	100,00	1	100,00
Img24	92	68,66	18	72,00	38	28,36	0	0,00	95	70,90	10	40,00
Img25	4	16,67	0	0,00	7	29,17	0	0,00	13	54,17	0	0,00
Img26	244	74,16	20	33,90	282	85,71	44	74,58	276	83,89	26	44,07
Img27	1	11,11	0	0,00	0	0,00	0	0,00	0	0,00	0	0,00
Img28	1	11,11	0	0,00	0	0,00	0	0,00	0	0,00	0	0,00
Img29	7	17,95	0	0,00	9	23,08	0	0,00	5	12,82	0	0,00
Img30	35	92,11	3	75,00	38	100,00	1	25,00	35	92,11	2	50,00

ANEXO VII

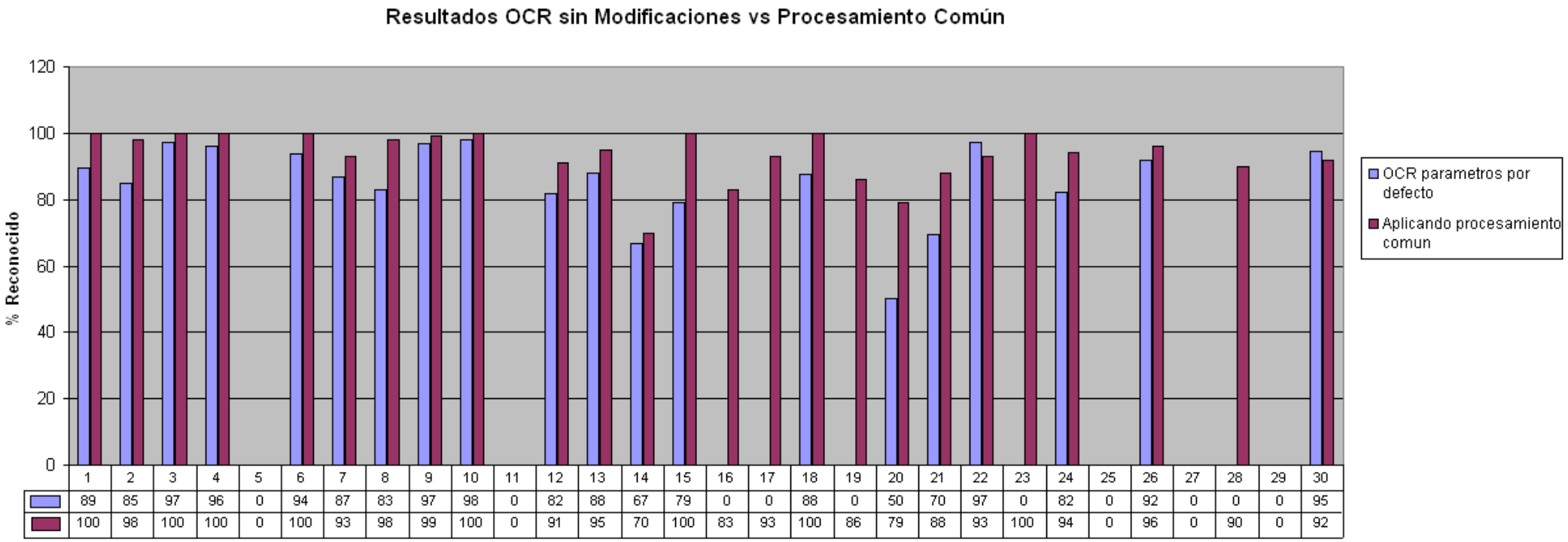

Resultados Pruebas Realizadas Etapa I
% Reconocido
90
80
70
60
50
40
30
20
10
0
de Prueba 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
%Reconocido 57 59 63 64 63 63 23 61 60 77 77 60 63 64 63 67 56
Pruebas:
1- OCR sin modificaciones
2- OCR modificación de sus parámetros
3- Canal R
4- Canal G
5- Canal B
6- Escala de grises
7- Canal S
8- Canal V
9- Remuestreo bicúbico 2X
10-Remuestreo bicúbico 4X
11-Remuestreo bicúbico 6X
12- Remuestreo a 300 ppi
13- Remuestreo 300 ppi + Canal R
14- Remuestreo 300 ppi + Canal R + Erosión 1
15- Remuestreo 300 ppi + Canal R + Erosión 2
16- Remuestreo 300 ppi + Canal R + Dilatación 1
17- Remuestreo 300 ppi + Canal R + Dilatación 2

ANEXO VIII



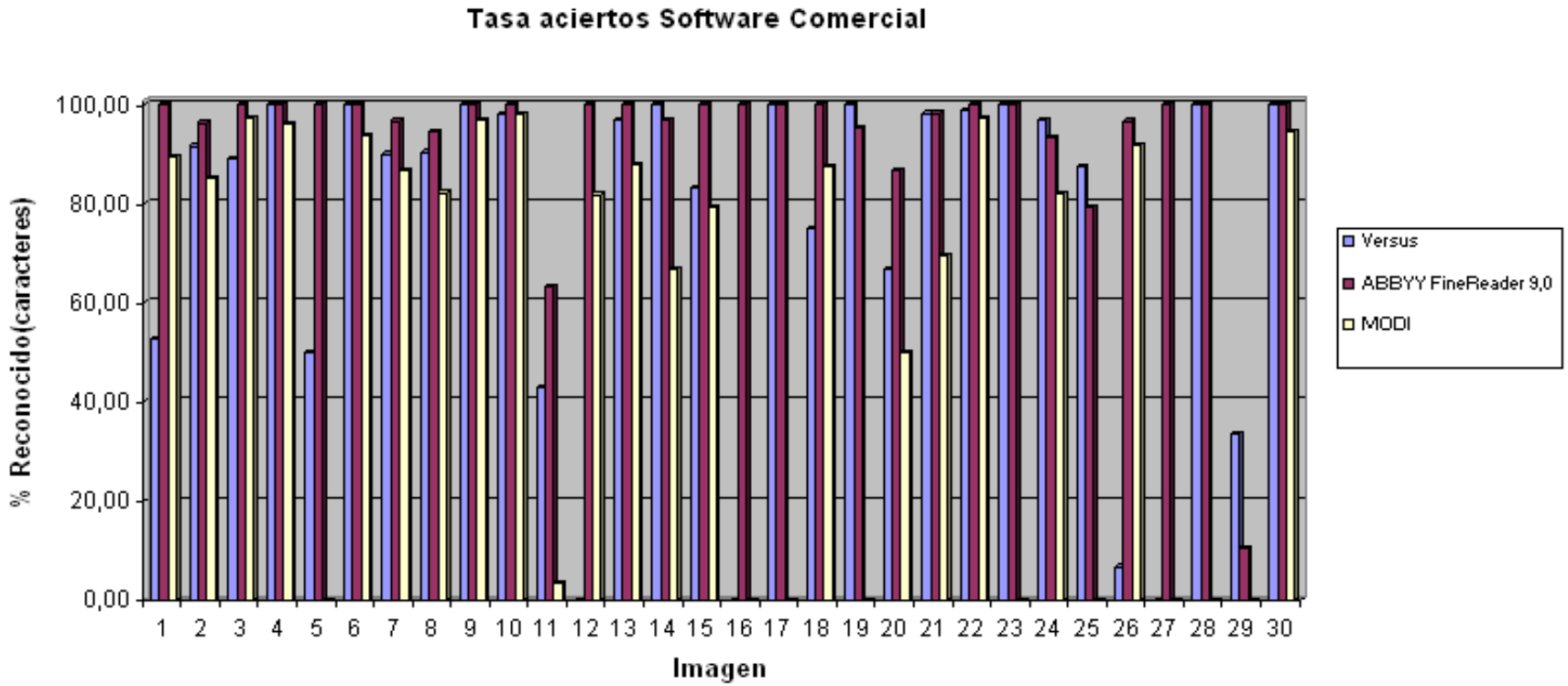
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
	100	98	100	100		100	93	98	100	100		100	99	97	100	94	96	100	95	79	96	100	100	94	91	96		90		100
	100	98	100	100	0	100	93	98	99	100	0	91	95	70	100	83	93	100	86	79	88	93	100	94	0	96	0	90	0	92

ANEXO IX

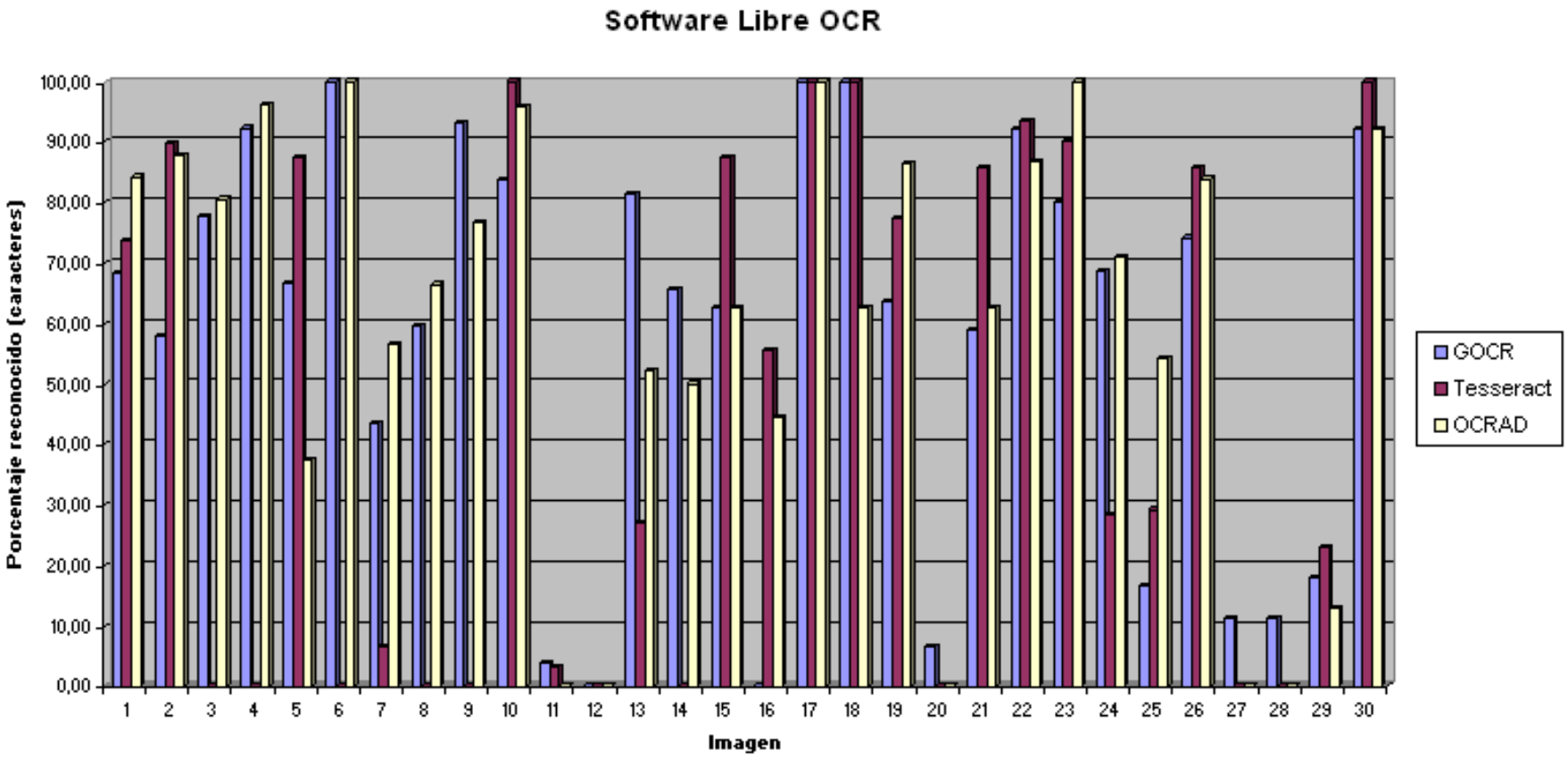


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
OCR parametros por defecto	89	85	97	96	0	94	87	83	97	98	0	82	88	67	79	0	0	88	0	50	70	97	0	82	0	92	0	0	0	95
Aplicando procesamiento comun	100	98	100	100	0	100	93	98	99	100	0	91	95	70	100	83	93	100	86	79	88	93	100	94	0	96	0	90	0	92

ANEXO X


Tasa aciertos Software Comercial
% Reconocido(caracteres)
100,00
80,00
60,00
40,00
20,00
0,00
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Imagen
Versus
ABBYY FineReader 9,0
MODI

ANEXO XI


Software Libre OCR
Porcentaje reconocido (caracteres)
100,00
90,00
80,00
70,00
60,00
50,00
40,00
30,00
20,00
10,00
0,00
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Imagen
GOCR
Tesseract
OCRAD

Anexo B. Criterios de clasificación manual de imágenes

Dentro del proyecto UML Models se está desarrollando un clasificador automático cuya función es la preselección de imágenes procedentes del Web.

Este clasificador se está construyendo mediante aprendizaje supervisado, es decir, a partir de ejemplos clasificados por humanos.

Con el fin de disponer de una muestra significativa de imágenes etiquetadas manualmente por expertos, los miembros del grupo han clasificado unas 18.000 imágenes con las siguientes categorías:

1. Imagen valida (diagrama valido)
2. Imagen no valida
3. Imagen dudosa (no apropiada para el aprendizaje)

Los porcentajes por tipo son:

1. 10%
2. 89%
3. 1%

La revisión de las imágenes así como la distribución de las etiqueta por experto ponen de manifiesto que los criterios se han aplicado de forma heterogénea.

OBJETIVOS:

- Homogenizar los criterios de clasificación manual de imágenes entre todas las personas involucradas en la tarea.
- Revisar con los nuevos criterios la muestra de 18.000 imágenes

CRITERIOS:

1. Las imágenes deben ser diagramas.
2. Las imágenes deben contener figuras semejantes a las utilizadas para representar clases, notas, componente, paquetes o puertos.
3. Las figuras deben contener información textual.
4. Las figuras deben estar enlazadas con otras.
5. Si se cumplen los puntos anteriores parcialmente o sólo en alguna zona de la imagen utilizar la opción dudosa
6. Si la representación no la producido una herramienta gráfica apropiada utilizar la opción dudosa.
7. Si la representación no está en dos dimensiones utilizar la opción dudosa.
8. Si se duda por cualquier otro motivo (elementos ornamentales solapamientos, etc.) utilizar la opción dudosa.

EJEMPLOS:

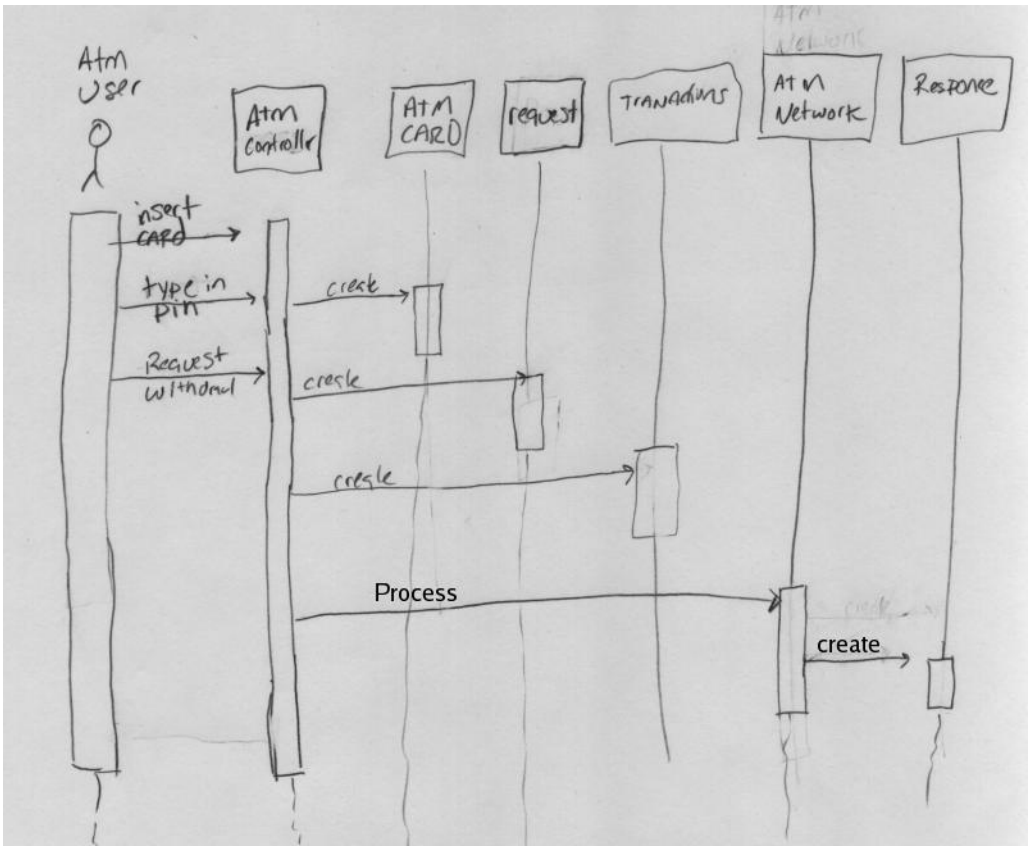


Ilustración B-1: Imagen no diseñada con herramientas gráficas apropiadas

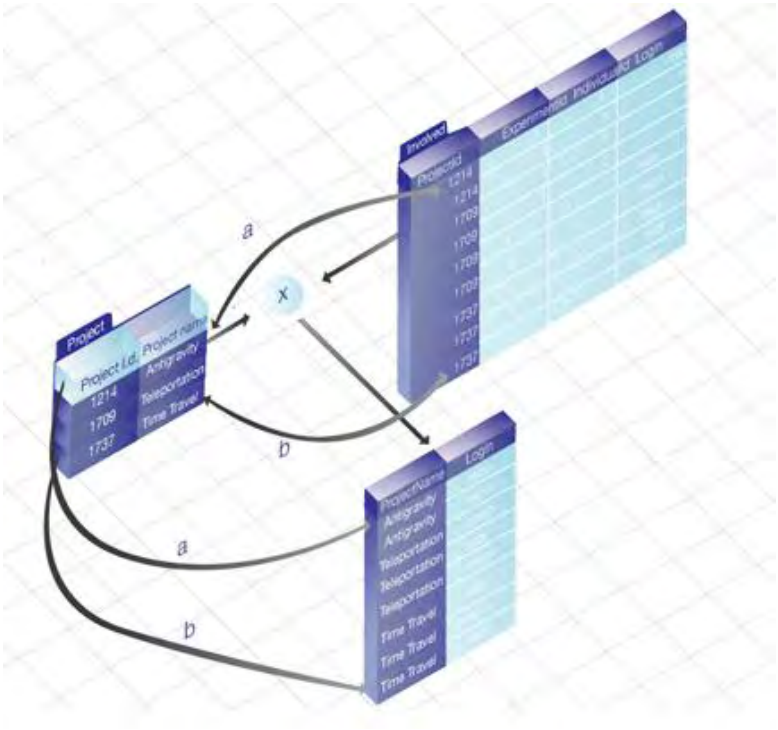


Ilustración B-2: Imagen en tres dimensiones

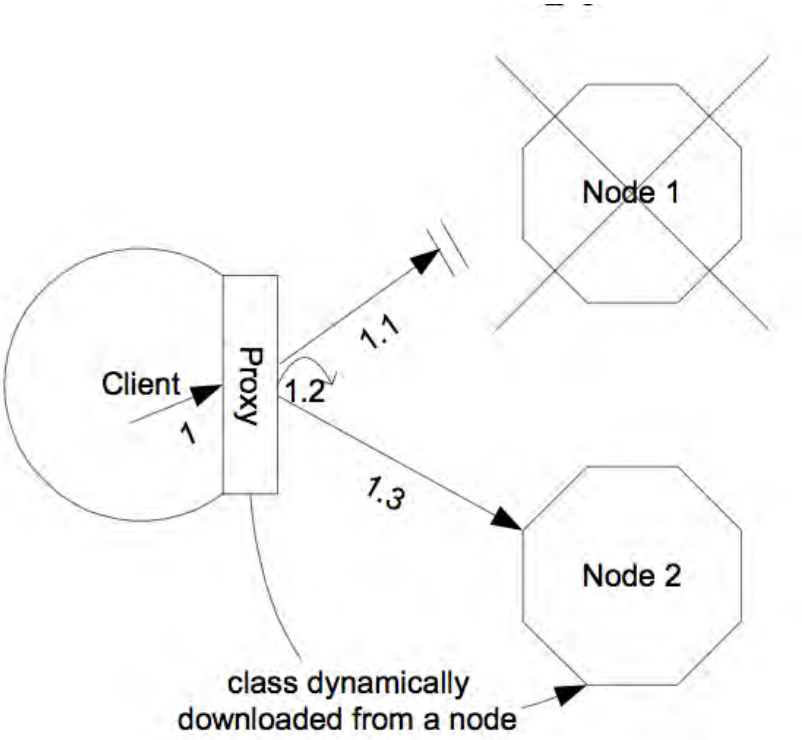


Ilustración B-3: Imagen con figuras inapropiadas

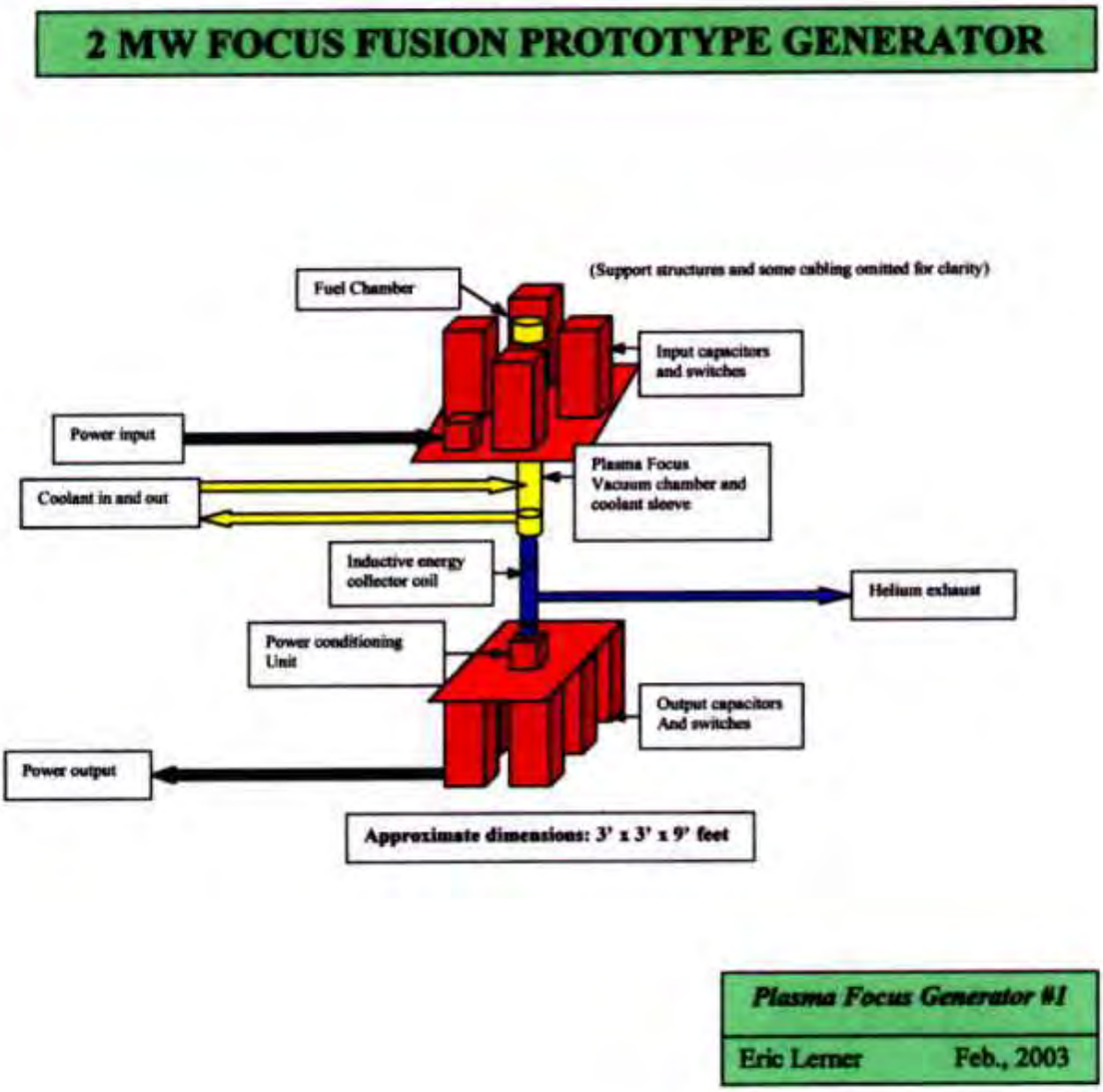


Ilustración B-4: Imagen valida parcialmente

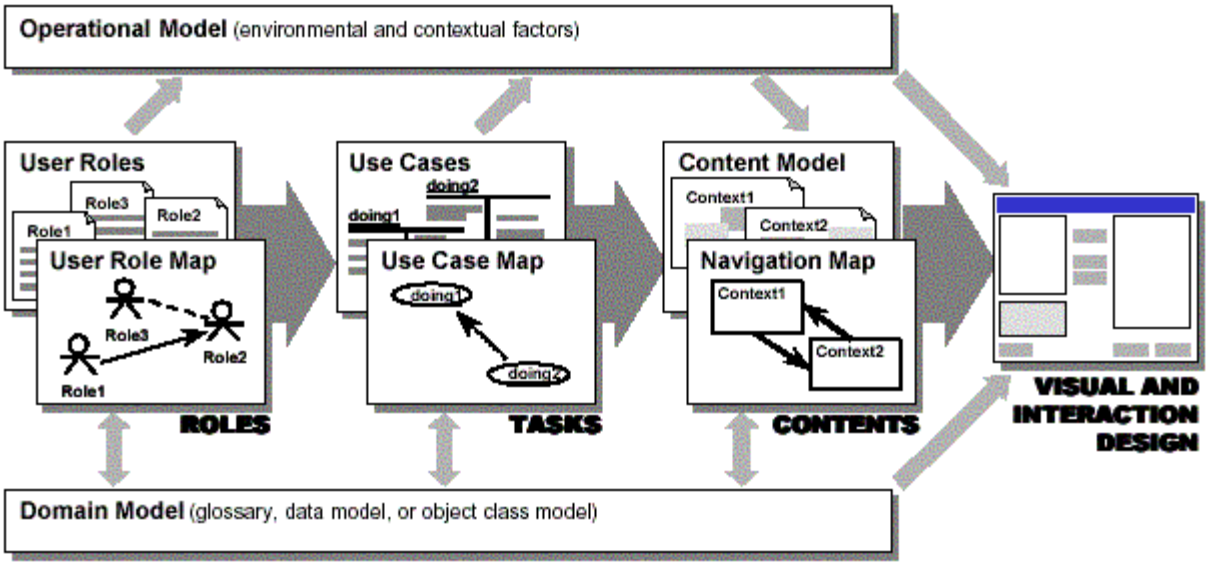


Ilustración B-5: Imagen con solapamientos

Figure 10 **ESS contextual view of customer access points. The customer may choose from a variety of different ways to access the financial aspects of the bank, but the supporting infrastructure should be common whenever possible.**

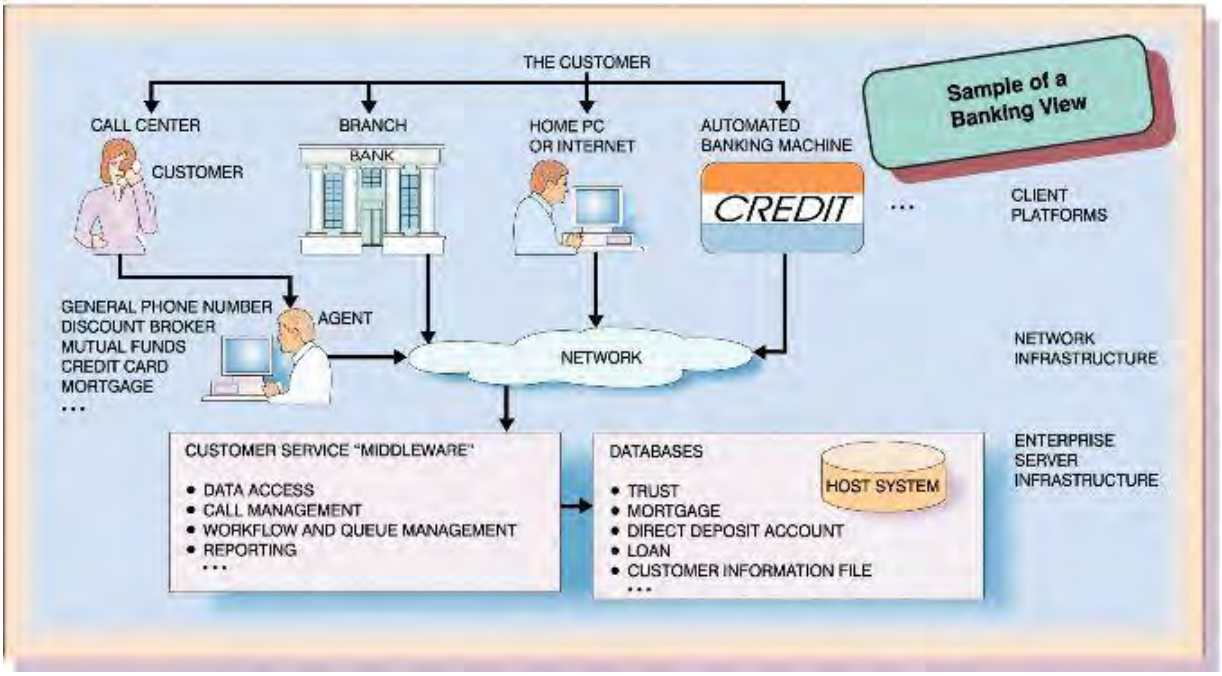


Ilustración B-6: Imagen valida parcialmente

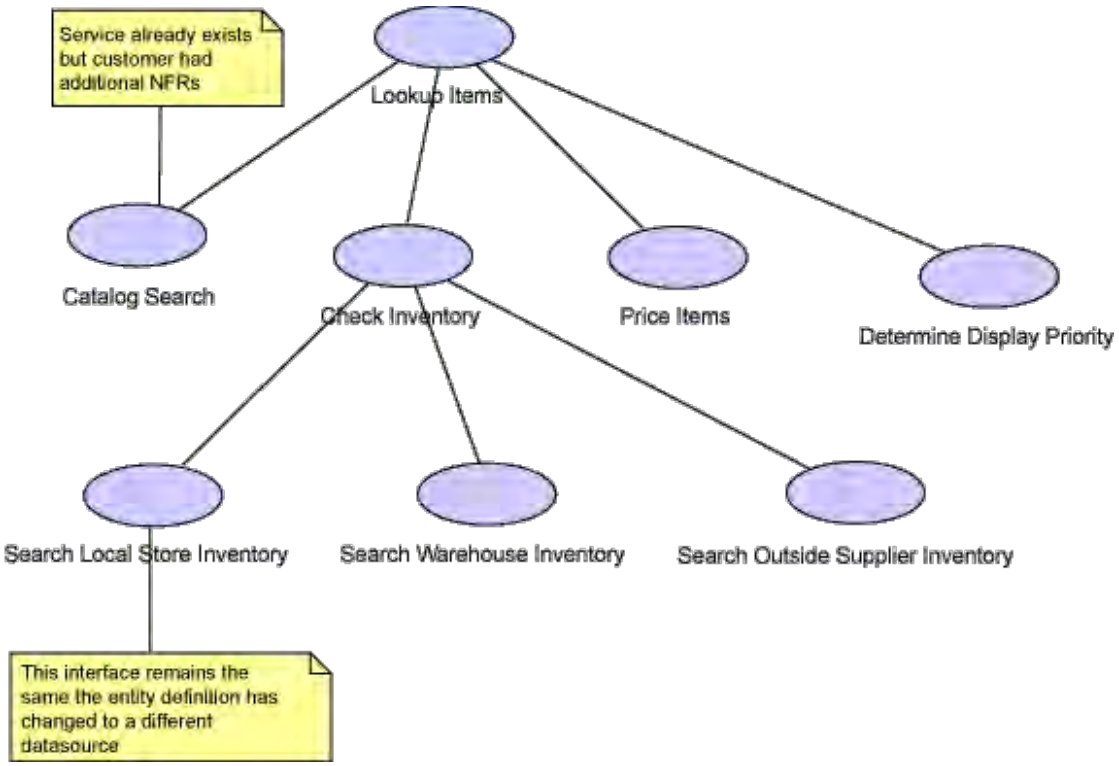


Ilustración B-7: Imagen valida parcialmente

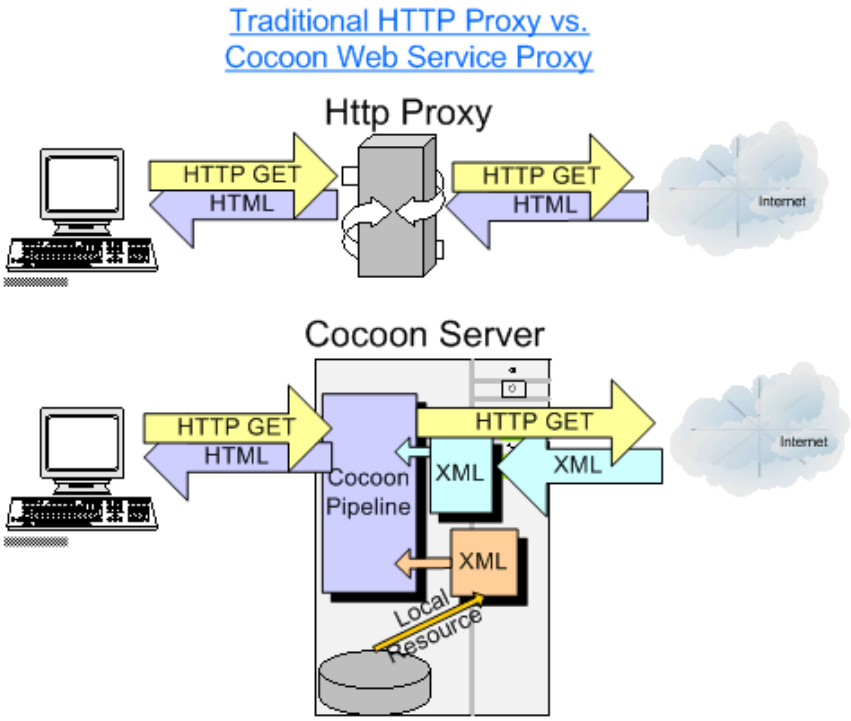


Ilustración B-8: Imagen dudosa

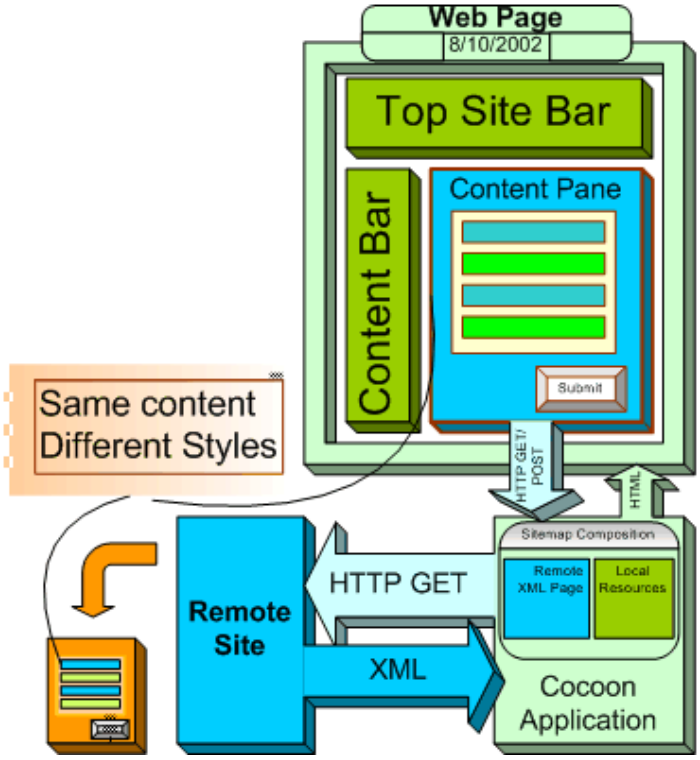


Ilustración B-9: Imagen dudosa

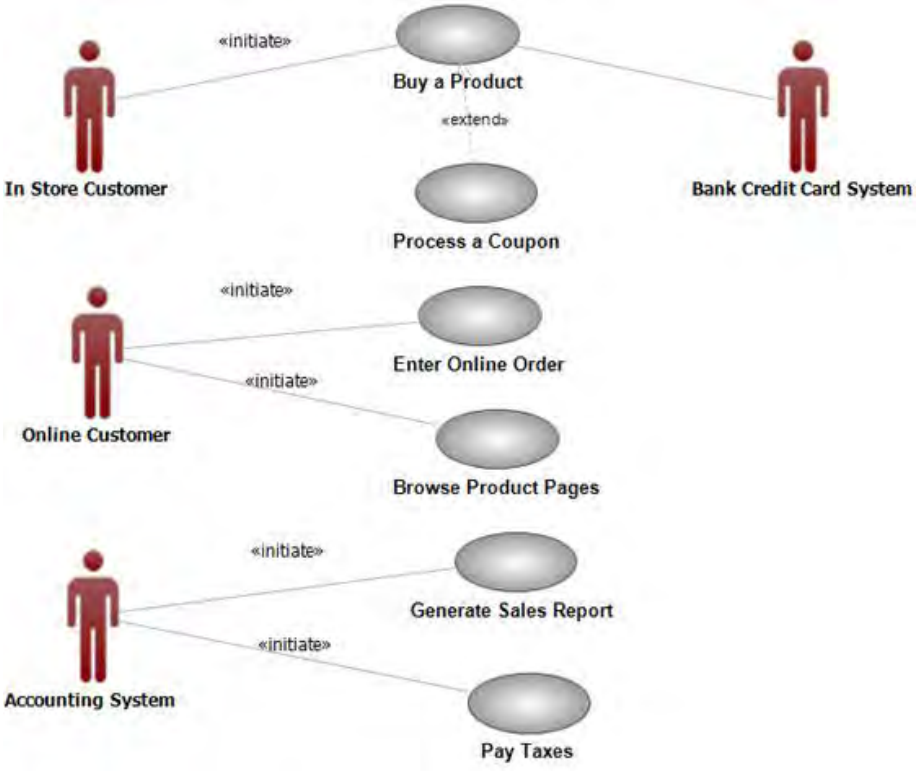


Ilustración B-10: Imagen carente de figuras objeto de estudio

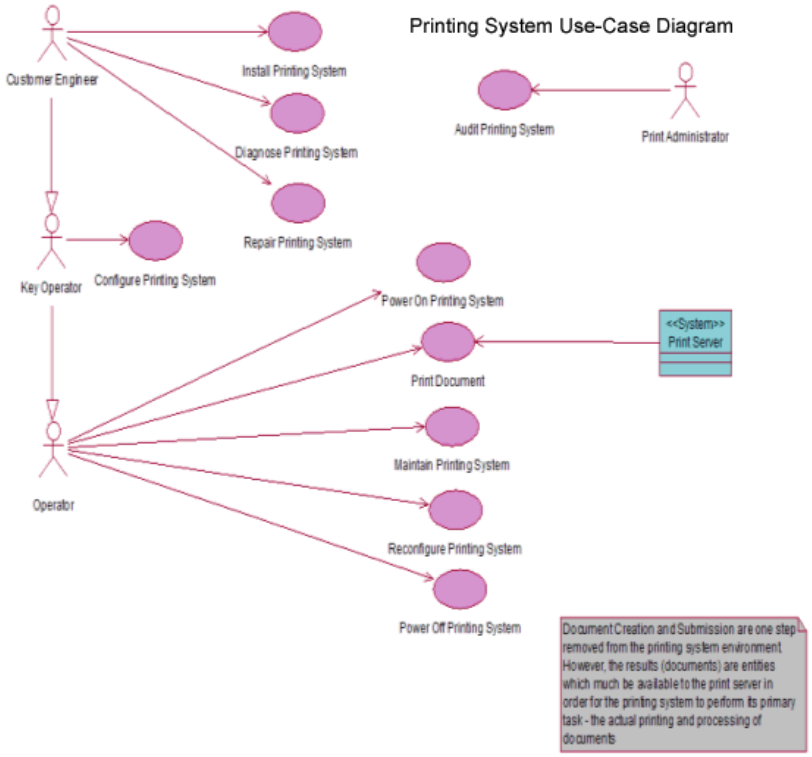


Ilustración B-11: Imagen valida parcialmente

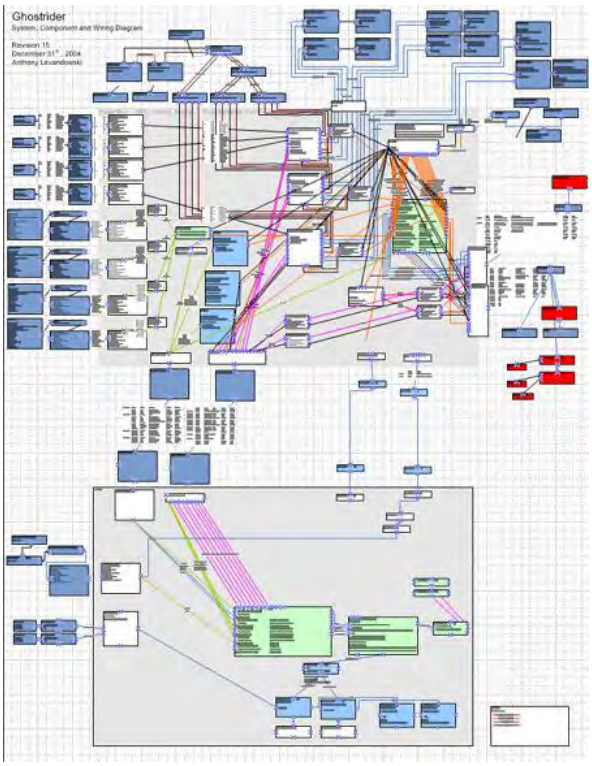


Ilustración B-12: Imagen dudosa

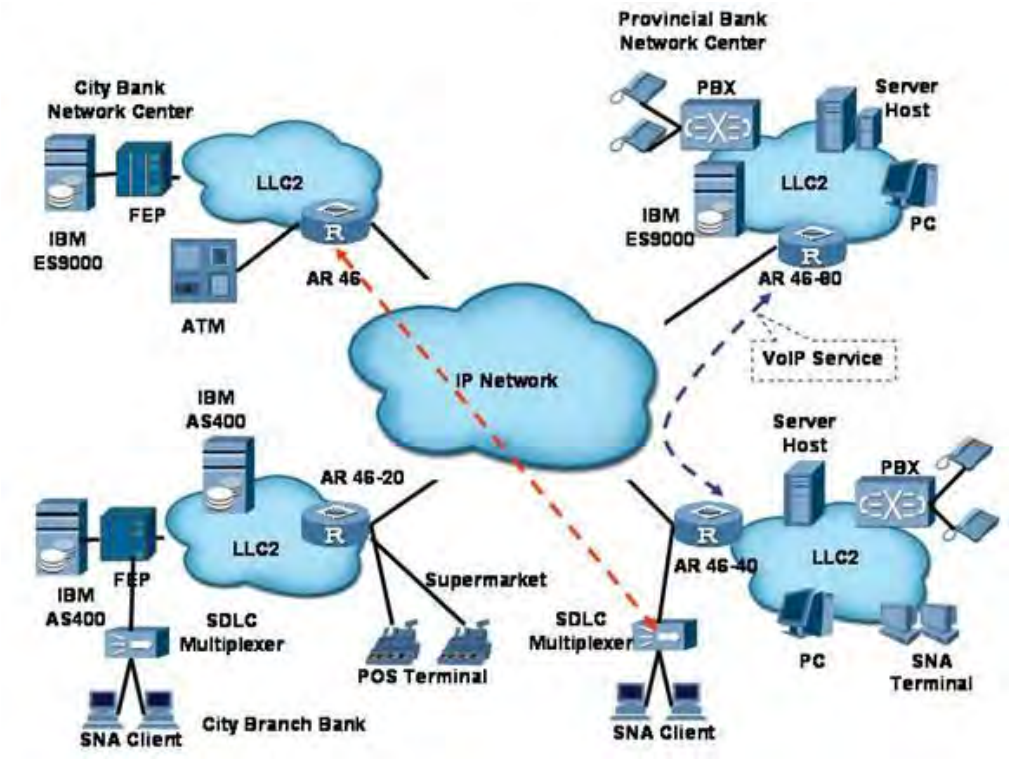


Ilustración B-13: Imagen carente de figuras objeto de estudio

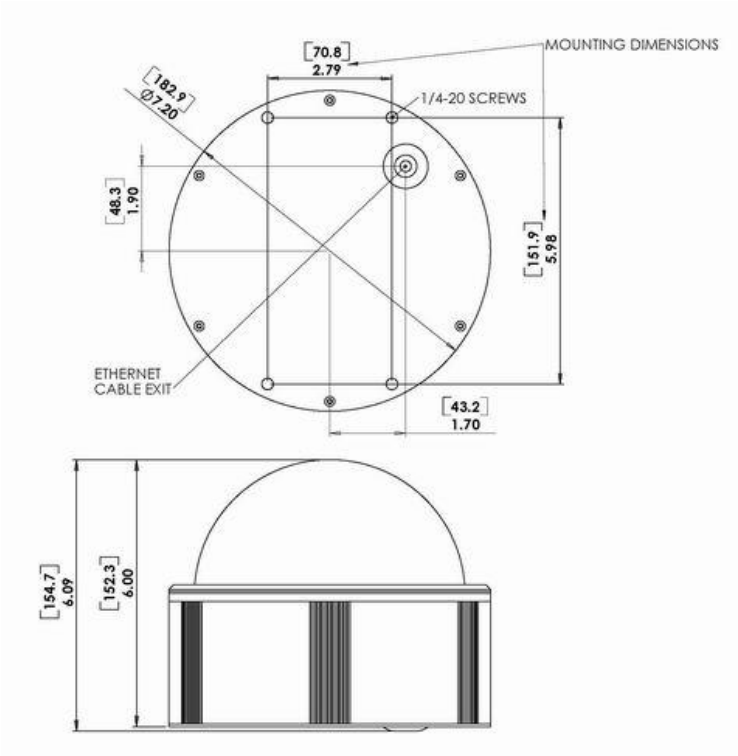


Ilustración B-14: Imagen sin información textual relevante

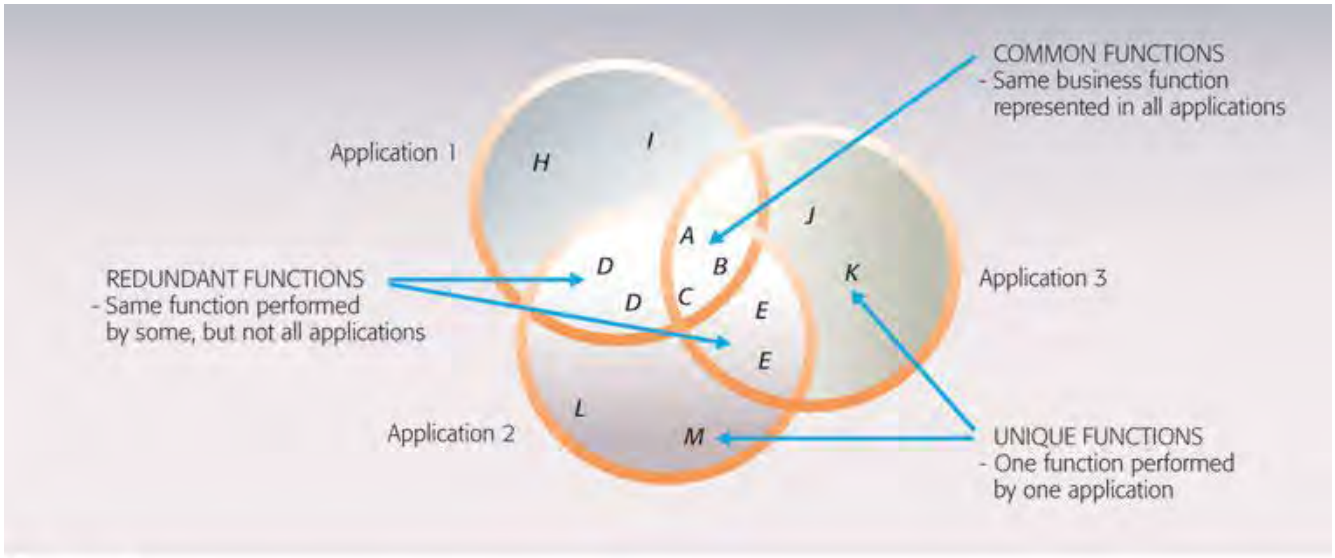


Figure 11
Venn diagram of legacy pattern at the application level

Ilustración B-15: Imagen con figuras inapropiadas

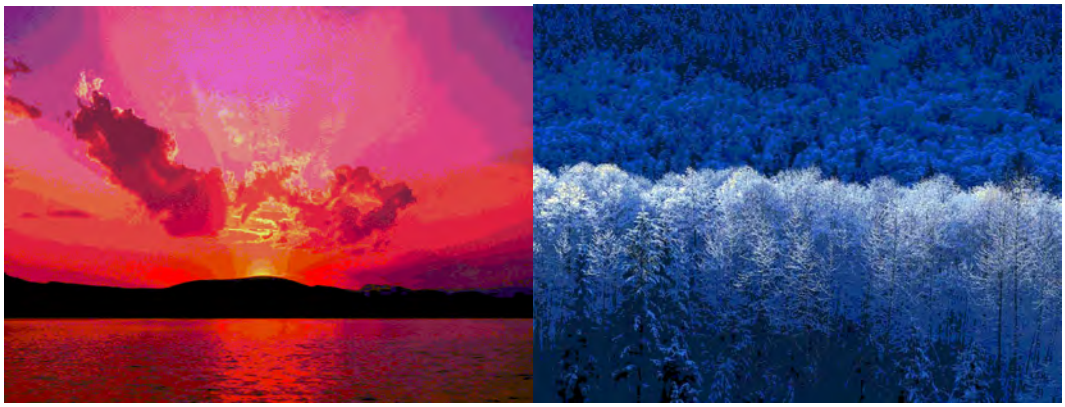


Ilustración B-16: Imágenes que no son diagramas

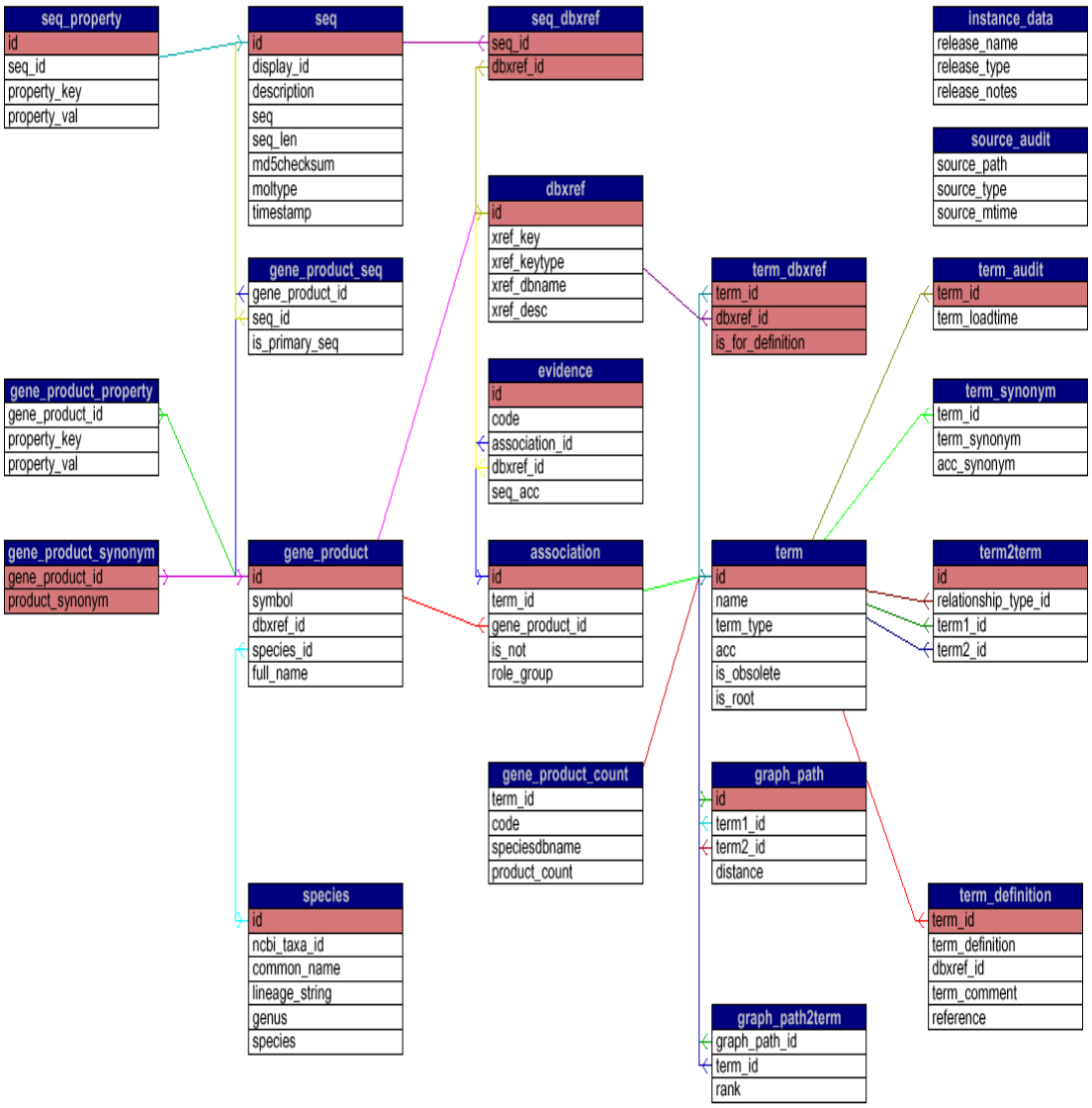


Ilustración B-17: Imagen que cumple los criterios

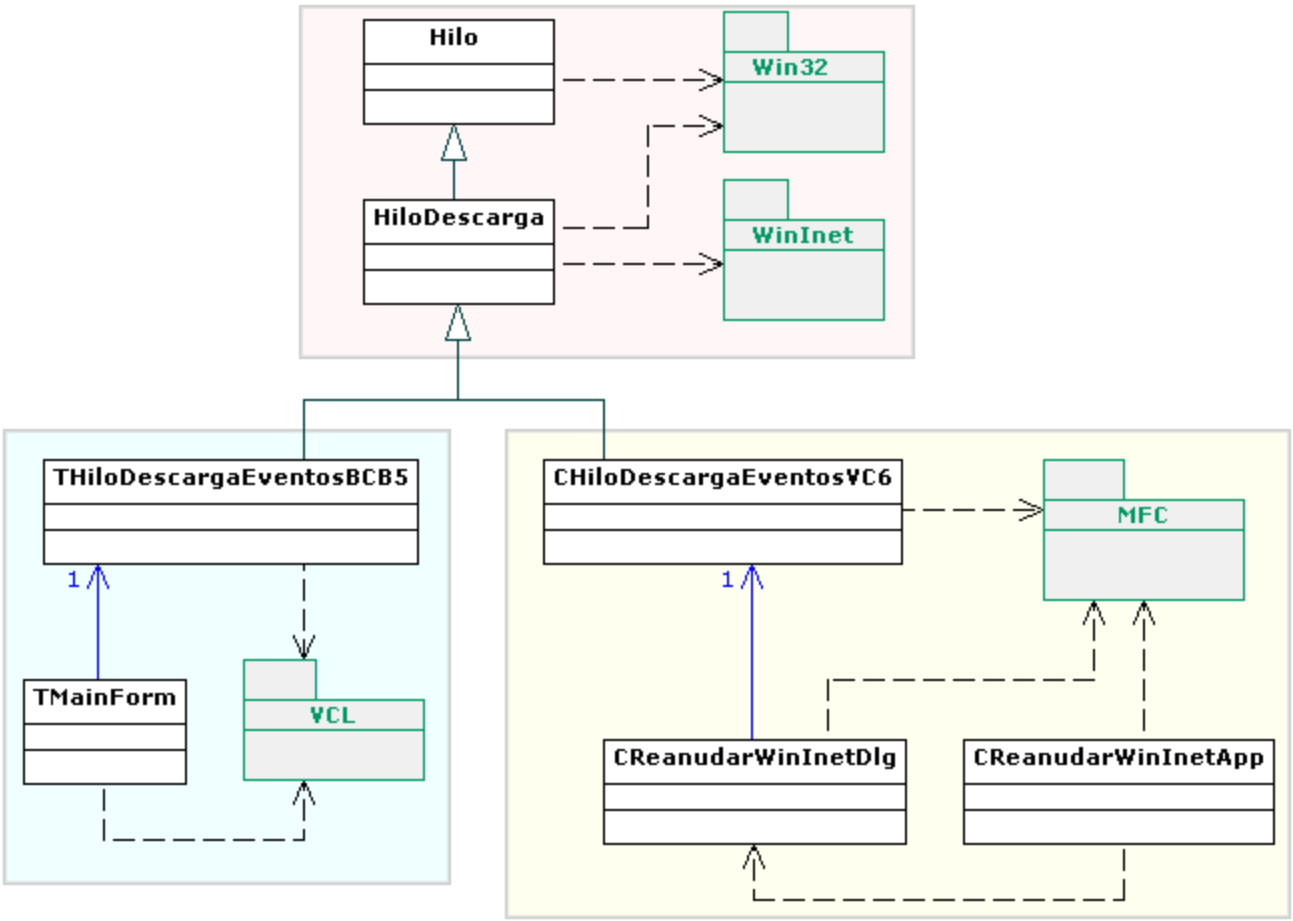


Ilustración B-18: Imagen que cumple los criterios

Anexo C. Ejemplos de procesados

Se muestran en este anexo cuatro ejemplos de procesados de imágenes que representan diagramas UML. Los procesados se explican paso a paso haciendo visible mediante ilustraciones cinco instantes de cada uno de los ejemplos.

Estas cinco ilustraciones se corresponden en cada ejecución a las siguientes situaciones:

1. **Imagen Inicial:** En las ilustraciones correspondientes a esta situación se muestran las imágenes que contienen los ejemplos de diagramas UML sin modificar. Es el punto inicial desde el que parte el procesado.
2. **Detección de líneas:** En esta situación sus ilustraciones asociadas presentan el resultado de la detección de todas las líneas horizontales y verticales presentes en las imágenes iniciales.
3. **Detección de objetos y enlaces:** La ilustraciones en este punto representan las formas reconocidas a parir de los segmentos previamente detectados. Se utiliza el código de color de la Tabla C-1 para distinguir mejor los resultados.

OBJETO	COLOR
Clase	
Nota	
Clase parametrizada	
Componente	

Paquete	
Enlaces	
Líneas Oblicuas	

Tabla C-1: Relación entre las formas UML y los colores que indican su reconocimiento.

4. **Limpieza de la imagen para la extracción del OCR:** Las representaciones de esta situación son replicas de las imágenes originales en las que se han eliminado los segmentos verticales pertenecientes a elementos gráficos. Este paso repercute favorablemente en el reconocimiento de texto llevado a cabo por el OCR.
5. **Remuestreo de las palabras del OCR:** Las ilustraciones de este instante del proceso contienen los listados de las palabras remuestreadas por el OCR según su tipo de letra con el objetivo de optimizar el porcentaje de texto extraído correctamente.

Para finalizar, la combinación de la información estructural y textual, encontrada en cada imagen, se indica para las relaciones de una forma intuitiva apta para comprobaciones y evaluaciones de resultados. Esta información se complementa con la de las formas representadas mediante el código de color. La estructura de la información relacional es la siguiente:

[identificador] (Nobre Forma) (Tipo relación y sentido) [identificador] (Nobre Forma)

Los identificadores previos a los nombres de formas son asignados en el proceso. Identifican las formas y permiten acceder a información relativa a tamaños y posiciones. No aportan información propia sobre los modelos UML representados.
Ejemplo de representación de la información:

[40]XNAAniSprite Sequence jerarquía |> [5]Enemy

En el ejemplo se indica una relación de jerarquía en el sentido indicado entre las formas *XNAAniSprite y Enemy.* La categoría de cada una quedarían asignadas por el color en que se representasen en las imágenes generadas en el punto 3. del proceso.

1. Procesado de imagen que representa clases

- **Imagen inicial**

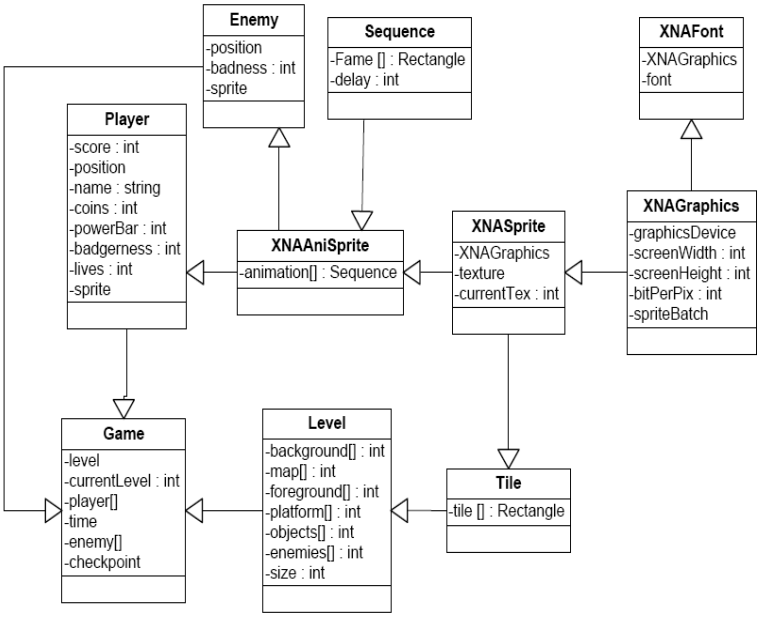


Ilustración C-1: Imagen inicial del ejemplo 1.

- **Detección de líneas**

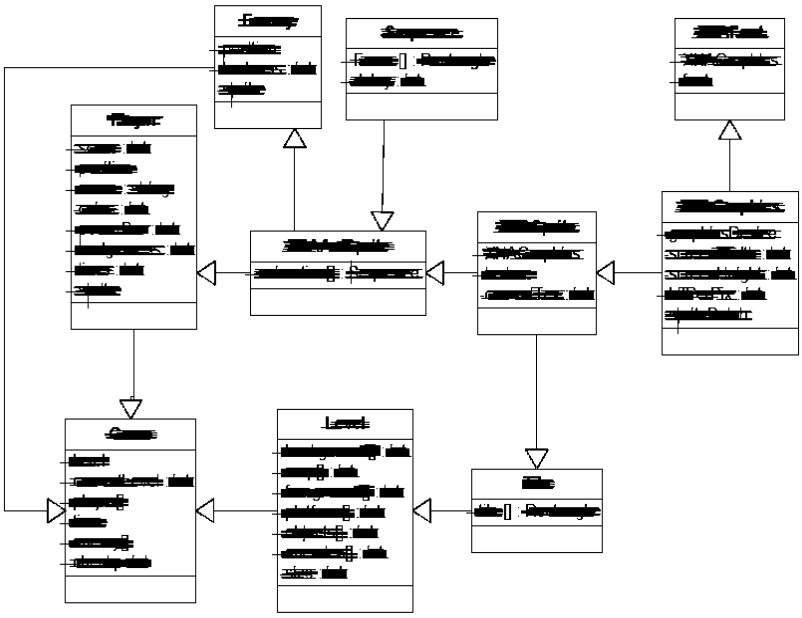


Ilustración C-2: Detección de líneas en el ejemplo 1.

- **Detección de objetos y de enlaces**

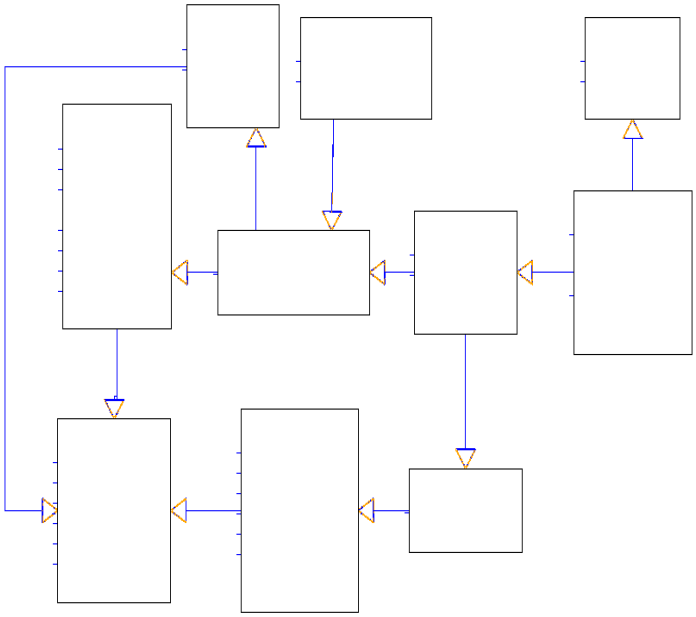


Ilustración C-3: Detección de formas y enlaces en el ejemplo 1.

- **Limpieza de la imagen para extracción del OCR**

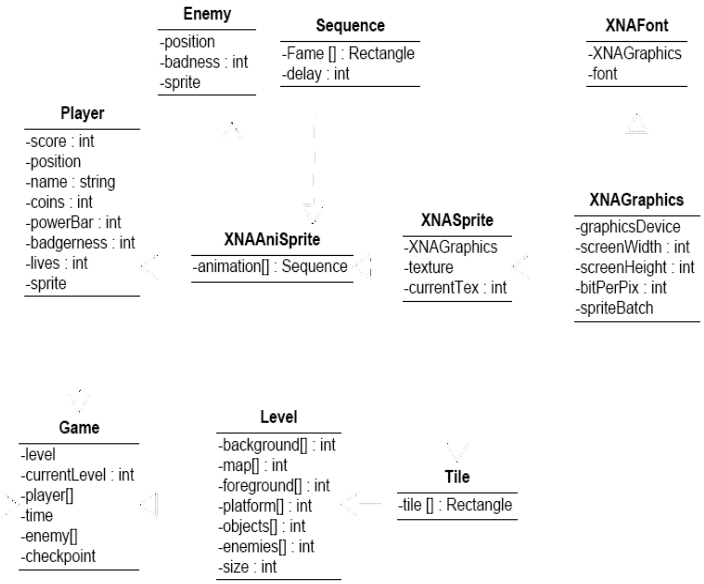


Ilustración C-4: Limpieza de imagen para la extracción de texto en el ejemplo 1.

- **Remuestreo de las palabras del OCR**

Ilustración C-5Remuestreo de palabras según su tipo de letra (ejemplo 1)

- **Salida**

[40]XNAAniSprite Sequence jerarquía |> [5]Enemy

[5]Enemy jerarquía |> [52]Game

[25]XNAGraphics jerarquía |> [14]XNAFont

[52]Game <| jerarquía [20]Player

[20]Player <| jerarquía [40]XNAAniSprite Sequence

[32]XNASprite <| jerarquía [25]XNAGraphics

[58]Tile <| jerarquía [32]XNASprite

[40]XNAAniSprite Sequence <| jerarquía [32]XNASprite

[48]Level <| jerarquia [58]Tile

[52]Game <| jerarquia [48]Level

2. Procesado de imagen que representa clases y paquetes

- **Imagen inicial**

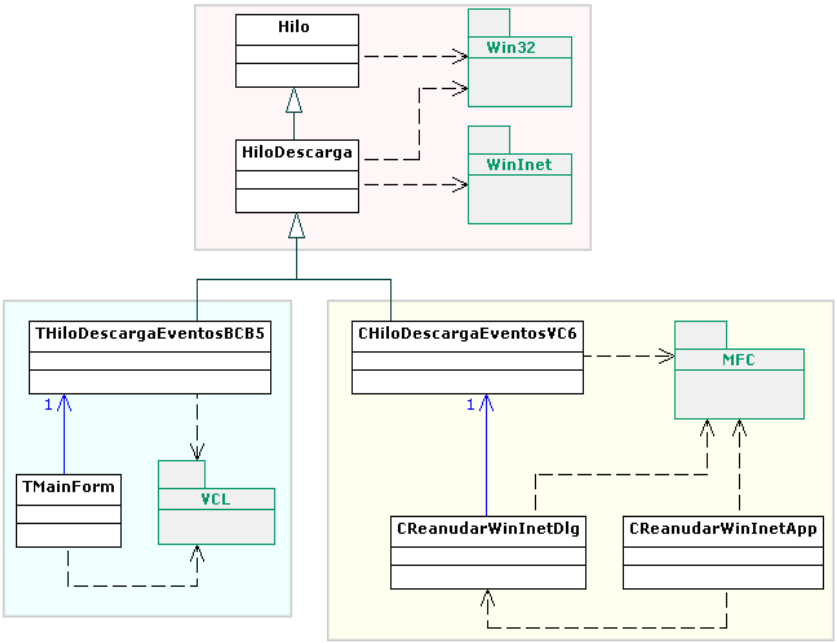


Ilustración C-6: Imagen inicial del ejemplo 2.

- **Detección de líneas**

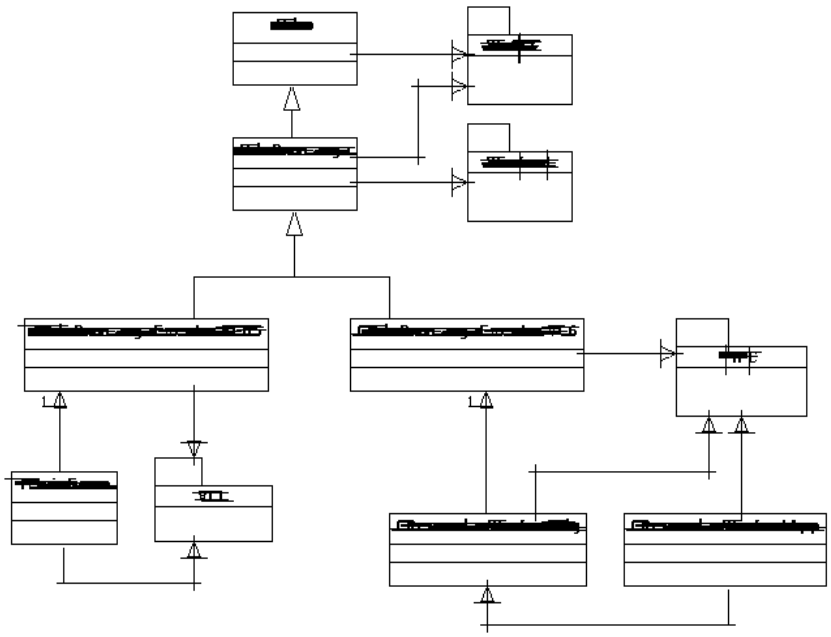


Ilustración C-7: Detección de líneas en el ejemplo 2.

- **Detección de objetos y de enlaces**

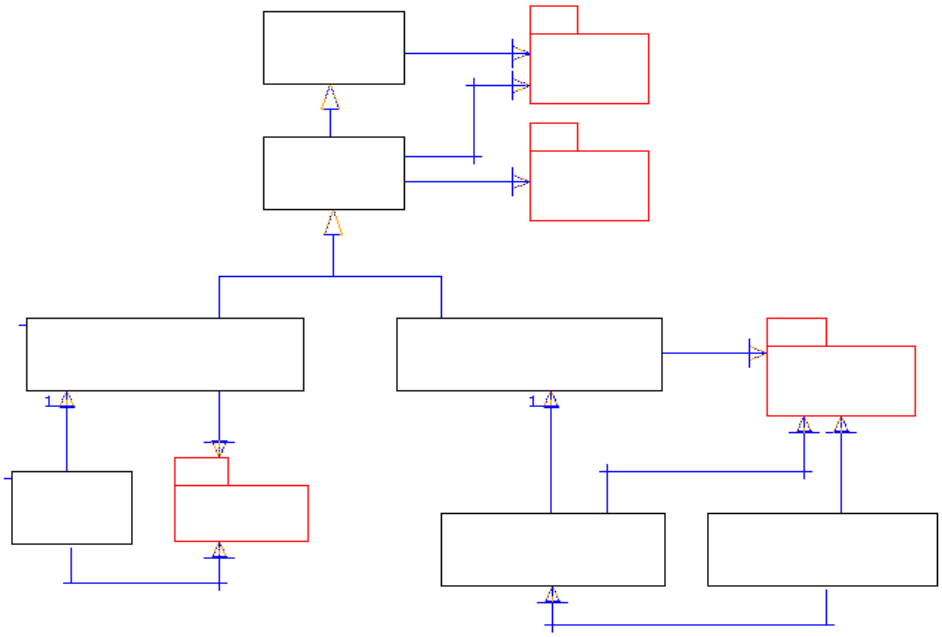


Ilustración C-8: Detección de formas y enlaces en el ejemplo 2.

- **Limpieza de la imagen para extracción del OCR**

Ilustración C-9: Limpieza de imagen para la extracción de texto en el ejemplo 2.

- **Remuestreo de las palabras del OCR**

Ilustración C-10: 11Remuestreo de palabras según su tipo de letra (ejemplo 2)

- **Salida**

[8]Hilo **jerarquía de interfaz** |> [2]Win32

[2]Win32 **< dependencia** [15]HiloDescarga

[15]HiloDescarga **dependencia >** [10]Winlnet

[29]C H ii oDes c a rg a Eve nto S V IC 6 **----------** [23]TH ii oDes c a rqa E ve

nto sB CB5 , **>** [15]HiloDescarga ;

[38]Trblai **asociativa >** [23]TH ii oDes c a rqa E vento sB CB5

[34]VL **< dependencia** [23]TH ii oDes c a rqa E vento sB CB5

[44]C Re an 'U d a rW ml **asociativa >** [29]C H ii oDes c a rg a Eve nto S V IC 6

[29]C H ii oDes c a rg a Eve nto S V IC 6 **dependencia >** [31]NFC

[34]VL **< dependencia** [38]Trblai

[44]C Re an 'Ud a rW ml **<| jerarquía de interfaz** [52]C Re an U d a rW 'mI netAp p

[44]C Re an 'U d a rW ml **dependencia >** [31]NFC

[52]C Re an U d a rW 'mI netAp p **jerarquía de interfaz |>** [31]NFC

3. Procesado de imagen que representa clases, clases parametrizadas y notas

- **Imagen inicial**

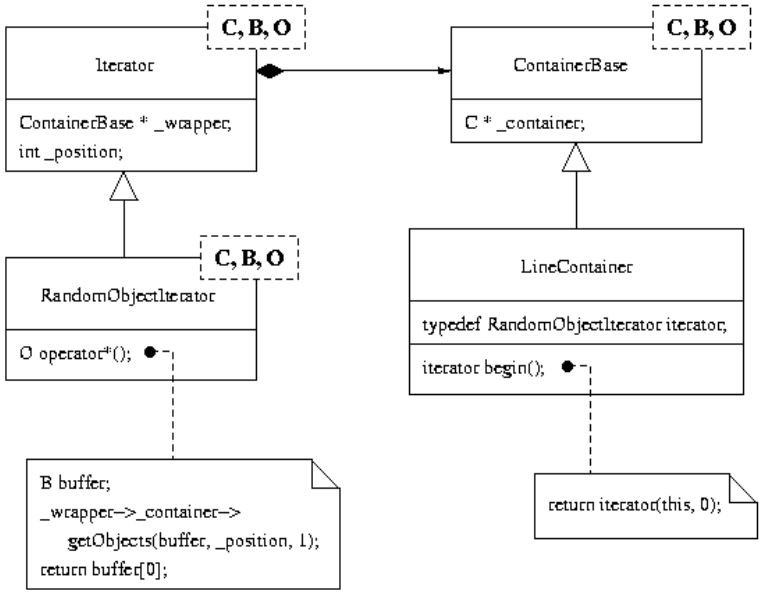


Ilustración C-12: Imagen inicial del ejemplo 3.

- **Detección de líneas**

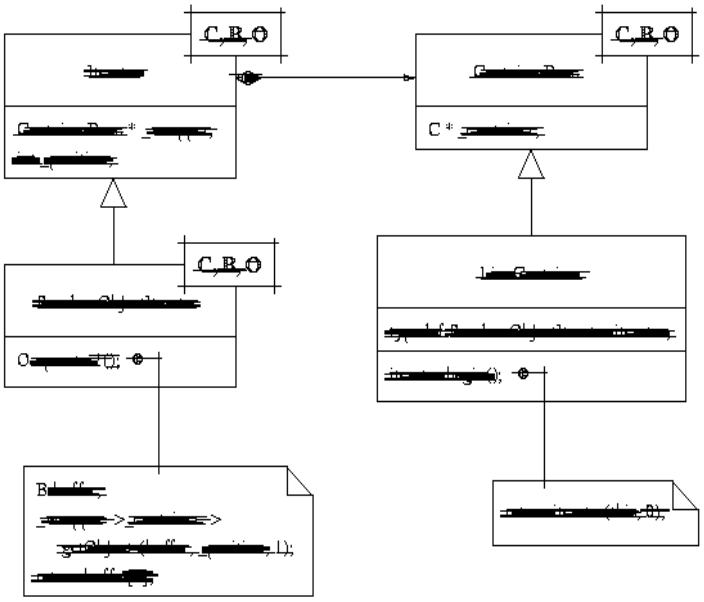


Ilustración C-13: Detección de líneas en el ejemplo 3.

- **Detección de objetos y de enlaces**

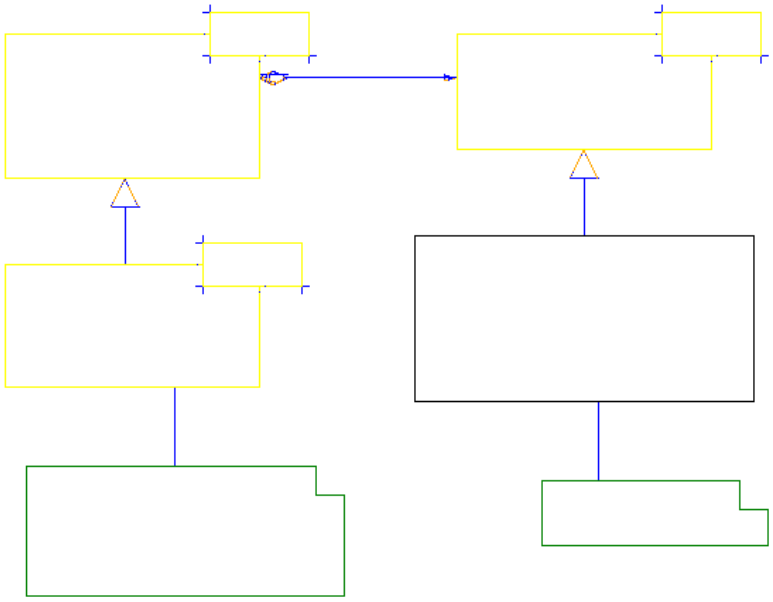


Ilustración C-14: Detección de formas y enlaces en el ejemplo 3.

- **Limpieza de la imagen para extracción del OCR**

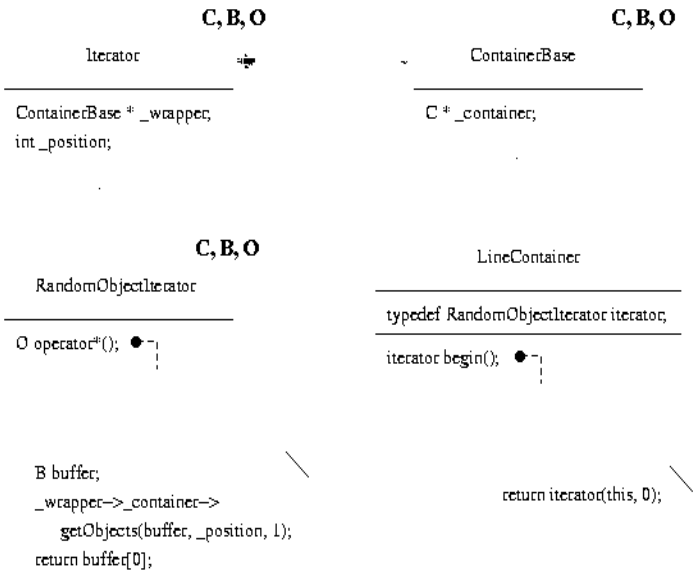


Ilustración C-15: Limpieza de imagen para la extracción de texto en el ejemplo 3.

- **Remuestreo de las palabras del OCR**

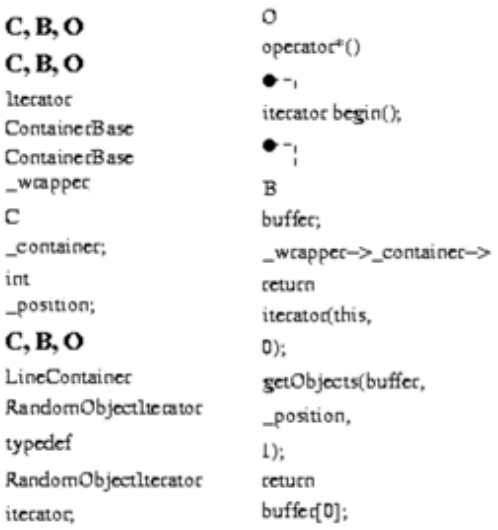


Ilustración C-16: 17Remuestreo de palabras según su tipo de letra (ejemplo 3)

- **Salida**

  [2]LineContainer **jerarquía** |> [30001]ContainerBase

  [30004]nota ---------- [2]LineContainer

  [30002]RandomObjectitecator **jerarquía** |> [30000]Iterator

  [30000]Iterator **jerarquía** |> [30000]Iterator

  [30000]Iterator <| **jerarquía** [30001]ContainerBase

  [30000]Iterator **jerarquía** |> [30000]Iterator

  [30003]nota ---------- [30002]RandomObjectitecator

4. Procesado de imagen que representa componentes

- **Imagen inicial**

  Esta imagen corresponde a un diagrama UML en el que solo aparecen componentes.

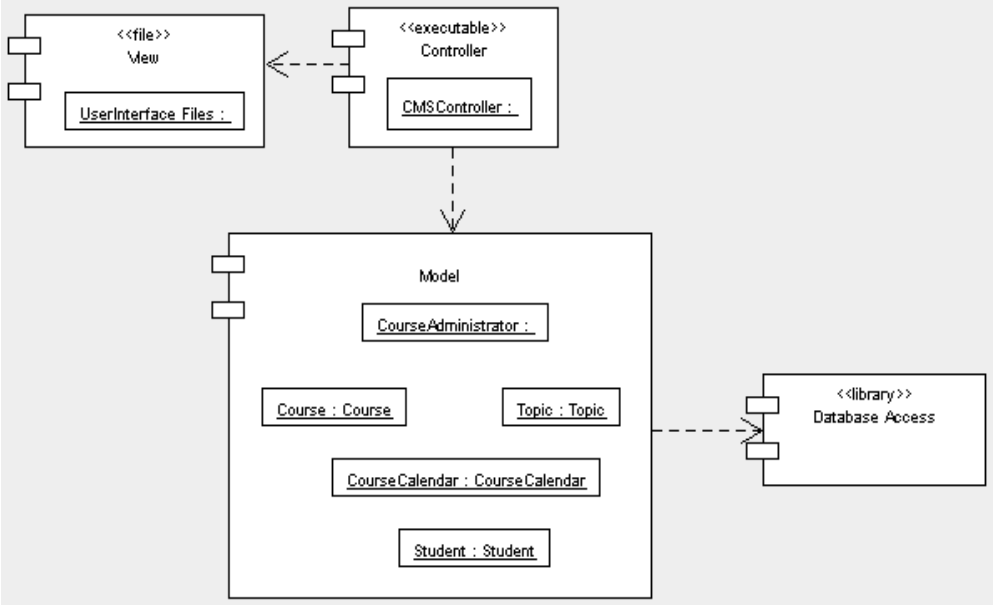


Ilustración C-18: Imagen inicial del ejemplo 4.

- **Detección de líneas**

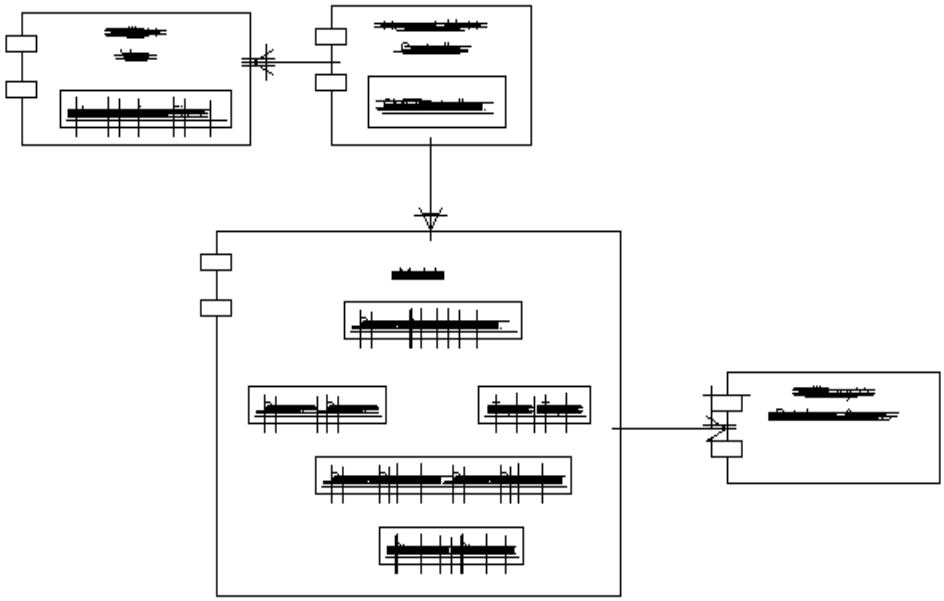


Ilustración C-19: Detección de líneas en el ejemplo 4.

- **Detección de objetos y de enlaces**

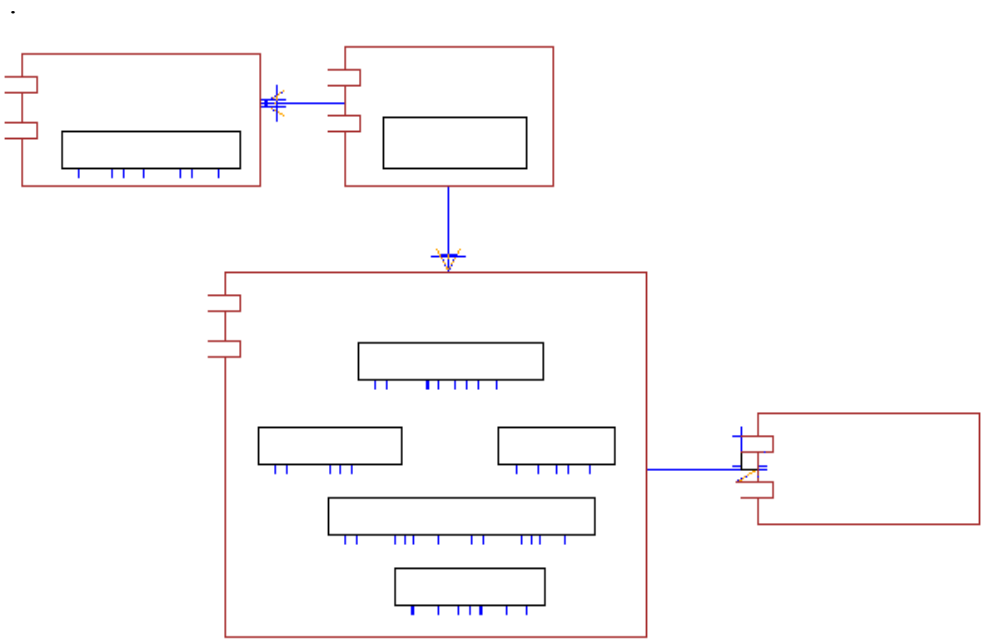


Ilustración C-20: Detección de objetos y enlaces en el ejemplo 4.

- **Limpieza de la imagen para extracción del OCR**

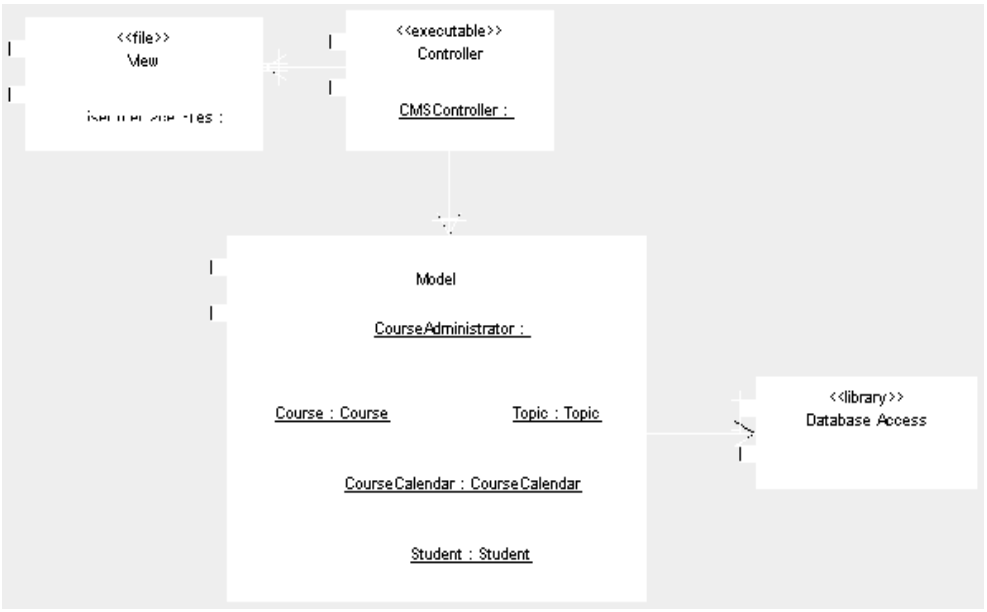


Ilustración C-21: Limpieza de imagen para la extracción de texto en el ejemplo 4.

- **Remuestreo de las palabras del OCR**

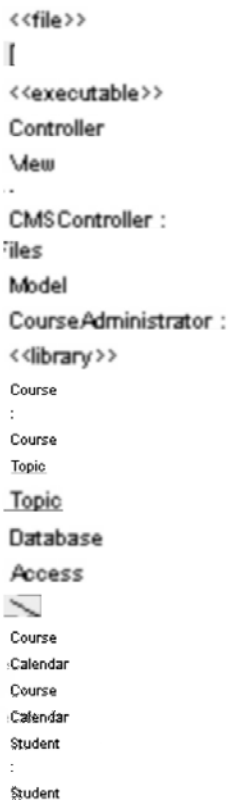


Ilustración C-22: 23Remuestreo de palabras según su tipo de letra (ejemplo 4)

- **Salida**

  [20003]<<library>> Database ?tcess **----------** [145]

  [20002]Model Course?ministrator: **<| jerarquía de interfaz**
  [20000]<<executable>

  > Controller

  [20001]<<file>> Userlnterface Files : **< dependencia**
  [20000]<<executable>> Con

  troller

  [20001]<<file>> Userlnterface Files : **dependencia >** [20001]<<file>>
  Userlnter face Files :

  [20001]<<file>> Userlnterface Files : **dependencia >** [20001]<<file>>
  Userlnter